MODULE-1

1. APPLICATIONS OF COMPUTER GRAPHICS:

1.1 Graphs and Charts

- An early application for computer graphics is the display of simple data graphs usually plotted on a character printer. Data plotting is still one of the most common graphics application.
- Graphs & charts are commonly used to summarize functional, statistical, mathematical, engineering and economic data for research reports, managerial summaries and other types of publications.
- Typically examples of data plots are line graphs, bar charts, pie charts, surface graphs, contour plots and other displays showing relationships between multiple parameters in two dimensions, three dimensions, or higher-dimensional spaces.
- Three dimensional graphs and charts are used to display additional parameter information, sometimes they are used for effect, providing more dramatic or more attractive presentations of the data relationships.

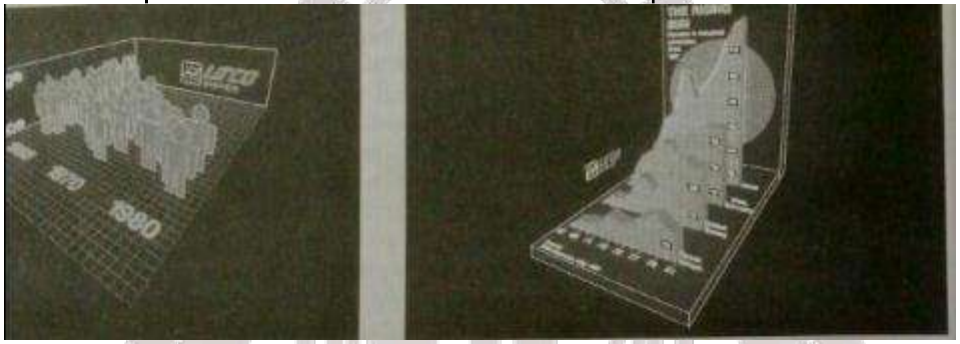


Fig: Two Three dimensional graphs designed for dramatic effect.

1.2 Computer-Aided Design

- A major use of computer graphics is in design processes-particularly for engineering and architectural systems.
- CAD, computer-aided design or CADD, computer-aided drafting and design methods are now routinely used in the automobiles, aircraft, spacecraft, computers, home appliances.
- Circuits and networks for communications, water supply or other utilities are constructed with repeated placement of a few geographical shapes.
- Animations are often used in CAD applications. Real-time, computer animations using wire-frame shapes are useful for quickly testing the performance of a vehicle or system.
- When object designs are complete, or nearly complete, realistic lighting conditions and surface rendering are applied to produce displays that will show the appearance of the final product.
- A circuit board layout, for example, can be transformed into a description of the individual processes needed to construct the electronics network.

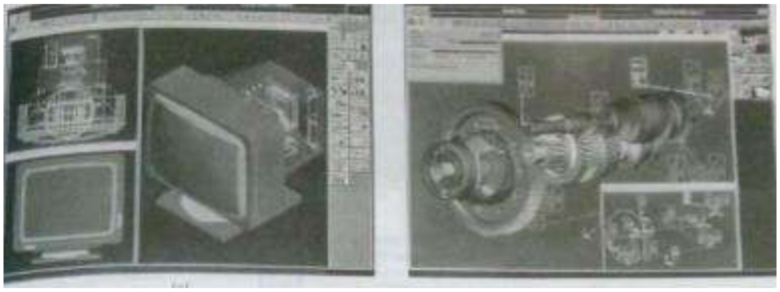


Fig: Multiple-window, color-coded CAD workstation displays.

1.3 Virtual-Reality Environments

- A more recent application of computer graphics is in the creation of virtual-reality environments in which a user can interact with the objects in a three dimensinal scene.
- Animations in virtual-reality environments are often used to train heavy-equipment operators or to analyze the effectiveness of various cabin configurations and control placements.
- With virtual-reality systems, designers and others can move about and interact with objects in various ways. Architectural designs can be examined by taking simulated "walk" through the rooms or around the outsides of buildings to better appreciate the overall effect of a particular design.
- With a special glove, we can even "grasp" objects in a scene and turn them over or move them from one place to another.

Fig: View of the tractor displayed on a standard monitor.

1.4 Data Visualizations:

- Producing graphical representations for scientific, engineering and medical data sets and processes is another fairly new application of computer graphics, which is generally referred to as **scientific visualization.** And the term **business visualization** is used in connection with data sets related to commerce, industry and other nonscientific areas.
- There are many different kinds of data sets and effective visualization schemes depend on the characteristics of the data. A collection of data can contain scalar values, vectors or higher-order tensors.

- Visual techniques are also used to aid in the understanding and analysis of complex processes and mathematical functions.
- A color plot of mathematical curve functions in fig a) and a surface plot of a function is shown in fig b).

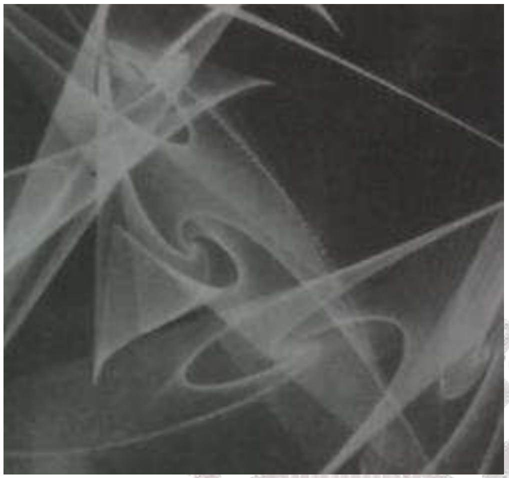


Fig a: Mathematical curve functions plotted in various color combinations.

Fig b: Lighting effects & surface-rendering techniques were applied to produce this surface representaion for a 3D function.

1.5 Education and Training

- Computer generated models of physical,financial,political,social,economic & other systems are often used as educational aids.
- Models of physical processes physiological functions,equipment, such as the color coded diagram as shown in the figure, can help trainees to understand the operation of a system.
- For some training applications,special hardware systems are designed.Examples of such specialized systems are the simulators for practice sessions ,aircraft pilots,air traffic- control personnel.
- Some simulators have no video screens,for eg: flight simulator with only a control panel for instrument flying.
- But most simulators provide screens for visual displays of the external environment.

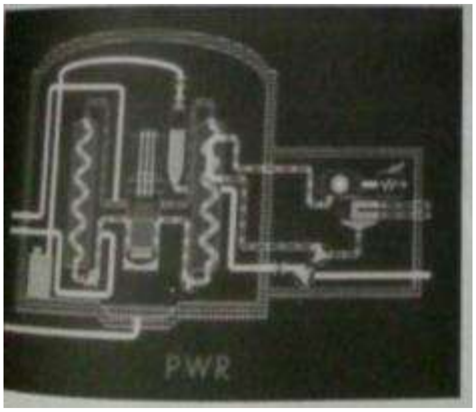


Fig: Color-coded diagram used to explain the operation of a nuclear reactor.

1.6 Computer Art

Fig: Cartoon drawing produced with a paint brush program, symbolically illustrating an artist at work on video monitor.

- Figure gives a figurative representation of the use of a **paintbrush program** that allows an artist to "paint" pictures on the screen of a video monitor.
  The picture is usually painted electronically on a graphics tablet using a stylus, which can simulate different brush strokes, brush widths and colors.
- Fine artists use a variety of other computer technologies to produce images. To create pictures the artist uses a combination of 3D modeling packages, texture mapping, drawing programs and CAD software etc.
- Commercial art also uses theses "painting" techniques for generating logos & other designs, page layouts combining text & graphics, TV advertising spots & other applications.
- A common graphics method employed in many television commercials is morphing, where one object is transformed into another.

1.7 Entertainment

- Television production, motion pictures, and music videos routinely a computer graphics methods.
- Sometimes graphics images are combined a live actors and scenes and sometimes the films are completely generated a computer rendering and animation techniques.
- Some television programs also use animation techniques to combine computer generated figures of people, animals, or cartoon characters with the actor in a scene or to transform an actor's face into another shape.
- Advanced computer-modeling & surface-rendering methods were employed in 2 award-winning short films to produce the scenes shown in the figure.

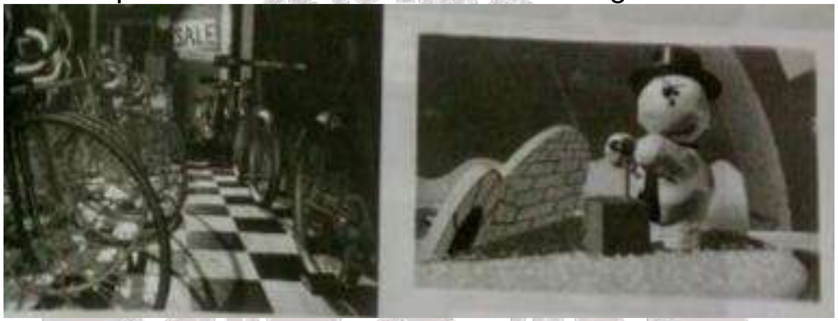


Fig: Computer-generated film scenes. a) Red's Dream b)Knickknack

1.8 Image Processing:

- The modification or interpretation of existing pictures, such as photographs and TV scans is called **image processing.**
- Methods used in computer graphics and image processing overlap, the two areas are concerned with fundamentally different operations.
- In computer graphics, a computer is used to create a pictures.
- Image processing methods are used to improve picture quality, analyze images, or recognize visual patterns for robotics applications.
- Image processing methods are often used in computer graphics, and computer graphics methods are frequently applied in image processing.
- The digital methods can be used to rearrange picture parts, to enhance color separation.
- Medical applications also make extensive use of image processing techniques for picture enhancements in tomography and in simulations and surgical operations.
- It is also used in computed X-ray tomography(CT), position emission tomography(PET),and computed axial tomography(CAT).

Fig: A blurred photograph of a license plate becomes legible after the application of image processing techniques.

1.9 Graphical User Interfaces

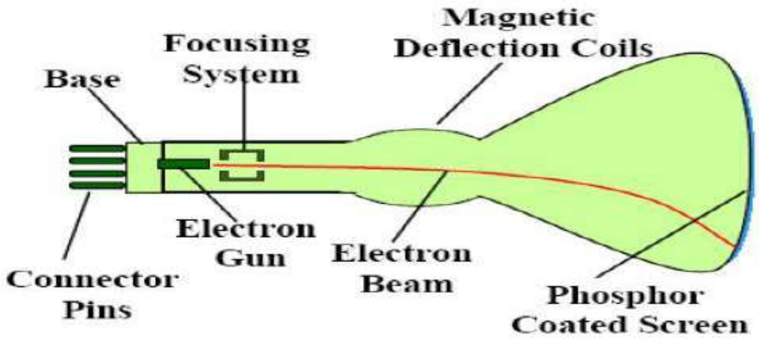
- It is common now for applications software to provide **graphical user interface** (GUI) .
- A major component of graphical interface is a window manager that allows a user to display multiple, rectangular screen areas called *display windows.*
- Each screen display area can contain a different process, showing graphical or non-graphical information, and various methods can be used to activate a display window.
- Using an interactive pointing device, such as mouse, we can active a display window on some systems by positioning the screen cursor within the window display area and pressing the left mouse button.

Fig: A graphical user interface, showing multiple display windows, menus, & icons.

2. **VIDEO DISPLAY DEVICES**:

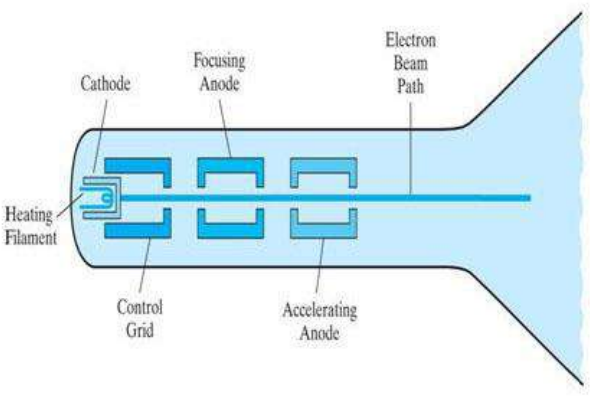
- The primary output device in a graphics system is a video monitor.
- The operation of most video monitors is based on the standard **cathode-ray tube (CRT)**.

2.1 Refresh Cathode-Ray Tubes:-



Basic design of a magnetic deflection CRT

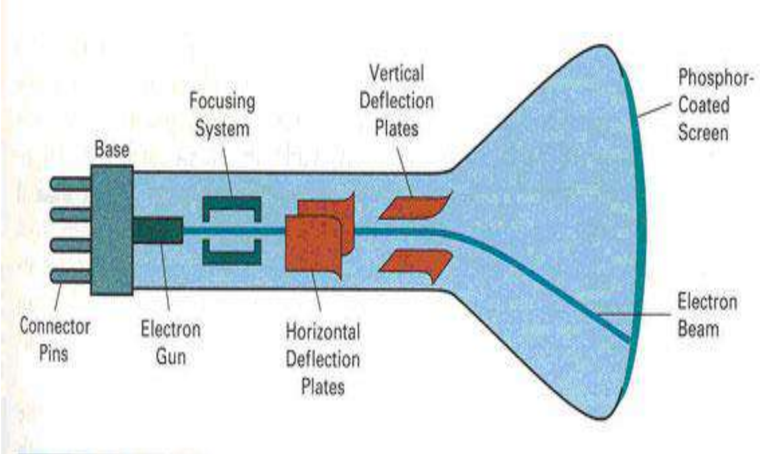
- A beam of electrons, emitted by an electron gun, passes through focusing and deflection systems that direct the beam toward specified positions on the phosphor-coated screen.
- The phosphor then emits a small spot of light at each position contacted by the electron beam and the light emitted by the phosphor fades very rapidly.
- One way to maintain the screen picture is to store the picture information as a charge distribution within the CRT in order to keep the phosphors activated.
- The most common method now employed for maintaining phosphor glow is to redraw the picture repeatedly by quickly directing the electron beam back over the same screen points. This type of display is called a **refresh CRT.**
- The frequency at which a picture is redrawn on the screen is referred to as the **refresh rate.**



Operation of an electron gun with an accelerating anode

- The primary components of an electron gun in a CRT are the heated metal cathode and a control grid.
- The heat is supplied to the cathode by directing a current through a coil of wire, called the filament, inside the cylindrical cathode structure.
- This causes electrons to be “boiled off” the hot cathode surface.
- Inside the CRT envelope, the free, negatively charged electrons are then accelerated toward the phosphor coating by a high positive voltage.
- Intensity of the electron beam is controlled by the voltage at the control grid.
- Since the amount of light emitted by the phosphor coating depends on the number of electrons striking the screen, the brightness of a display point is controlled by varying the voltage on the control grid.
- The focusing system in a CRT forces the electron beam to converge to a small cross section as it strikes the phosphor and it is accomplished with either electric or magnetic fields.
- With electrostatic focusing, the electron beam is passed through a positively charged metal cylinder so that electrons along the center line of the cylinder are in equilibrium position.
- Deflection of the electron beam can be controlled with either electric or magnetic fields.
- Cathode-ray tubes are commonly constructed with two pairs of magnetic-deflection coils.
- One pair is mounted on the top and bottom of the CRT neck, and the other pair is mounted on opposite sides of the neck.
- The magnetic field produced by each pair of coils results in a traverse deflection force that is perpendicular to both the direction of the magnetic field and the direction of travel of the electron beam.

- Horizontal and vertical deflections are accomplished with these pair of coils.

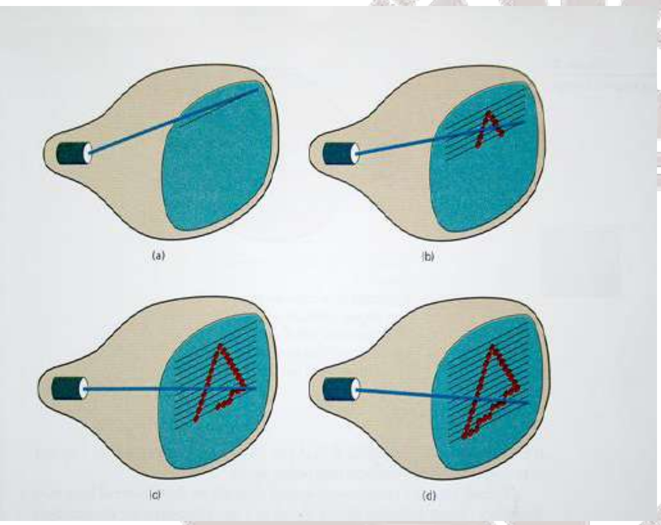


Electrostatic deflection of the electron beam in a CRT

- When electrostatic deflection is used, two pairs of parallel plates are mounted inside the CRT envelope where, one pair of plates is mounted horizontally to control vertical deflection, and the other pair is mounted vertically to control horizontal deflection.
- Spots of light are produced on the screen by the transfer of the CRT beam energy to the phosphor.
- When the electrons in the beam collide with the phosphor coating, they are stopped and their kinetic energy is absorbed by the phosphor.
- Part of the beam energy is converted by the friction in to the heat energy, and the remainder causes electros in the phosphor atoms to move up to higher quantum-energy levels.
- After a short time, the “excited” phosphor electrons begin dropping back to their stable ground state, giving up their extra energy as small quantum of light energy called photons.
- What we see on the screen is the combined effect of all the electrons light emissions: a glowing spot that quickly fades after all the excited phosphor electrons have returned to their ground energy level.
- The frequency of the light emitted by the phosphor is proportional to the energy difference between the excited quantum state and the ground state.
- Lower persistence phosphors required higher refresh rates to maintain a picture on the screen without flicker.
- The maximum number of points that can be displayed without overlap on a CRT is referred to as a **resolution.**
- Resolution of a CRT is dependent on the type of phosphor, the intensity to be displayed, and the focusing and deflection systems.
- High-resolution systems are often referred to as high-definition systems.

2.2 Raster-Scan Displays:-

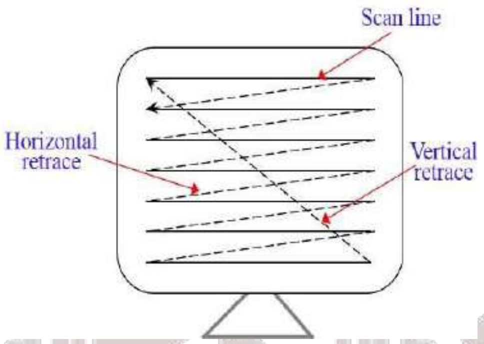
- The most common type of graphics monitor employing CRT is the **raster-scan display**, based on television technology.
- In a raster-scan system, the electron beam is swept across the screen, one row at a time, from top to bottom, where each row is referred to as a **scan line.**
- As the electron beam moves across a scan line, the beam intensity is turned on and off to create a pattern of illuminated spots.
- Picture definition is stored in a memory area called the **refresh buffer** or **frame buffer**, where **frame** refers to the total screen area.
- The memory area holds the set of color values for the screen points. These stored color values are then retrieved from the refresh buffer and used to control the intensity of the electron beam as it moves from spot to spot across the screen.
- The refresh buffer is used to store the set of screen color values, it is also called as **color buffer.**



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

- Each screen spot that can be illuminated by the electron beam is referred to as a **pixel** or **pel** (**picture element**).
- Raster systems are commonly characterized by their resolution, which is the number of pixel positions that can be plotted.
- Another property of video monitors is **aspect ratio,** which is often defined as the number of pixel columns divided by the number of scan lines that can be displayed by the system.
- Aspect ratio can also be described as the number of horizontal points to vertical points (or vice versa).
- The number of bits per pixel in a frame buffer is referred to as either the **depth** of the buffer area or the number of **bit planes.**
- A frame buffer with one bit per pixel is called as **bitmap,** and a frame buffer with multiple bits per pixel is called as **pixmap.**
- The terms bitmap and pixmap are also used to describe other rectangular arrays, where a bitmap is any pattern of binary values and a pixmap is a multicolor pattern.

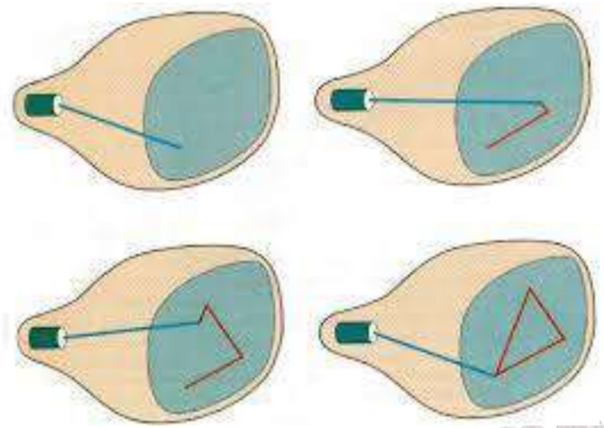
- As each screen refresh takes place, we tend to see each frame as a smooth continuation of the patterns in the previous frame, as long as the refresh rate is not too low.
- Refresh rates are described in units of cycles per second, or Hertz (Hz), where a cycle corresponds to one frame.
- At the end of each scan line, the electron beam returns to the left side of the screen to begin displaying the next scan line.
- The return to the left of the screen, after refreshing each scan line, is called the **horizontal retrace** of the electron beam.
- At the end of each frame, the electron beam returns to the top left corner of the screen (**vertical retrace**) to begin the next frame.



- On some raster-scan systems and TV sets, each frame is displayed in two passes using an interlaced refresh procedure.
- In the first pass, the beam sweeps across every other scan line from top to bottom.
- After the vertical retrace, the beam then sweeps out the remaining scan lines.
- Interlacing of the scan lines in this way allows us to see the entire screen displayed in one-half the time it would have taken to sweep across all the lines at once from top to bottom.
- This technique is primarily used with slow refresh rates.
- This is an effective technique for avoiding flicker-provided that adjacent scan lines contains similar display information.

2.3 Random-Scan Displays:-

- In a **random-scan display** unit, a CRT has the electron beam directed only to those parts of the screen where a picture is to be displayed.
- Pictures are generated as line drawings, with the electron beam tracing out the component lines one after the other. Hence random-scan monitors are also referred to as **vector displays** or **stroke-writing displays** or **calligraphic displays.**

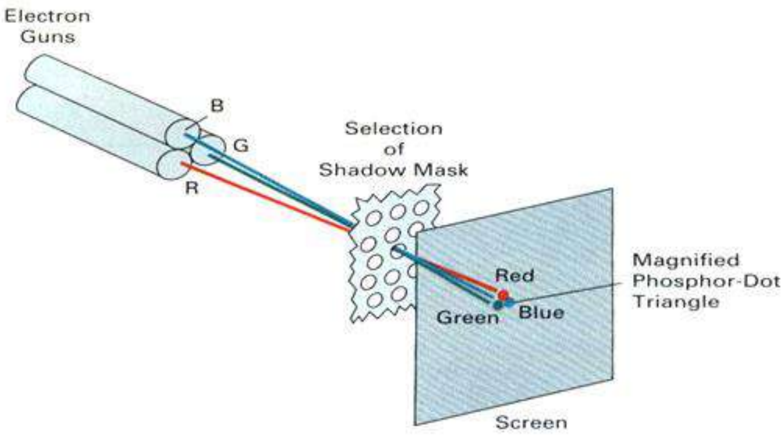


A random-scan system draws the component lines of an object in any specified order

- The component lines of a picture can be drawn and refreshed by a random-scan system in any specified order.
- Refresh rate on a random-scan system depends on the number of lines to be displayed on that system.
- Picture definition is now stored as a set of line-drawing commands in an area of memory referred to as the **display list, refresh display file, vector file,** or **display program.**
- To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn.
- After all line-drawing commands have been processed, the system cycles back to the first line command in the first list.
- When a small set of lines is to be displayed, each refresh cycle is delayed to avoid very high refresh rates, which could burn out the phosphor.
- Since picture definition is stored as a set of line-drawing instructions rather than as a set of intensity values for all screen points, vector displays generally have higher resolutions than raster systems.

2.4 Color CRT Monitors:

- A CRT monitor displays color pictures by using a combination of phosphors that emit different-colored light.
- **Beam-penetration** method uses only two phosphor layers: red and green.
- A beam of slow electrons excites only the outer red layer, but a beam of very fast electrons penetrates through the red layer and excites the inner green layer.
- At intermediate beam speeds, combinations of red and green light are emitted to show two additional colors, orange and yellow.
- **Shadow-mask** methods are commonly used in raster-scan systems since they produce a much wider range of colors than the beam penetration method.



Operation of a delta-delta, shadow-mask CRT

- This approach is based on the way that we seem to perceive colors as combinations of red, green, and blue components, called the **RGB color model.**
- Thus, a shadow-mask CRT uses three phosphor color dots at each pixel position. One phosphor dot emits a red light, another emits a green light, and the third emits a blue light.
- This type of CRT has three electron guns, one for each color dot, and a shadow-mask grid just behind the phosphor-coated screen.
- The light emitted from the three phosphors results in a small spot of color at each pixel position, since our eyes tend to merge the light emitted from the three dots into one composite color.
- In the figure above, three electron beams are deflected and focused as a group onto the shadow mask, which contains a series of hoes aligned with the phosphor-dot patterns.
- 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.
- 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.
- Another configuration for the three electron guns is an in-line arrangement in which the three electron guns, and the corresponding red-green-blue color dots on the screen, are aligned along one scan line instead of in a triangular pattern.
- We obtain color variations in a shadow-mask CRT by varying the intensity levels of the three electron beams (red, green, or blue).
- When all three dots are activated with equal beam intensities, white color is obtained.
- Yellow is produced with equal intensities from the green and red dots.
- Magenta is produced with equal blue and red intensities and cyan when blue and green are activated equally and more sophisticated systems can allow intermediate intensity levels, so that several million colors are possible.
- Color graphics systems can be used with several types of CRT display devices.
- Color CRTs in graphics systems are designed as **RGB monitors**. These monitors use shadow-mask methods and take the intensity level for each electron gun directly from the computer system without any intermediate processing.

- An RGB color system with 24 bits of storage per pixel is generally referred to as a **full-color system** or a **true-color system.**

3.RASTER-SCAN SYSTEMS:

Interactive raster-graphics system typically employ several processing units. In addition to the central processing unit, or CPU, a special-purpose processor, called the video controller or display controller, is used to control the operation of the display device. Organization of a simple raster system is shown as below.

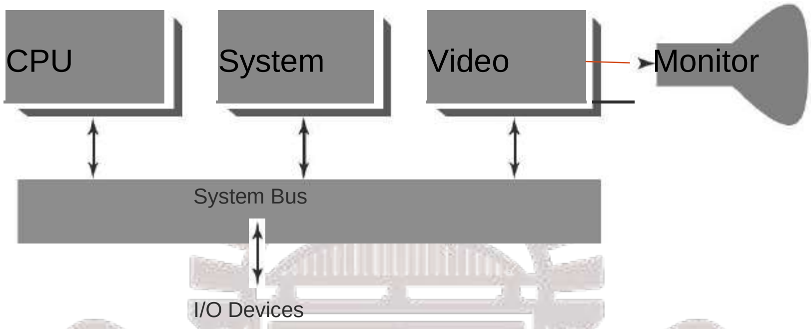
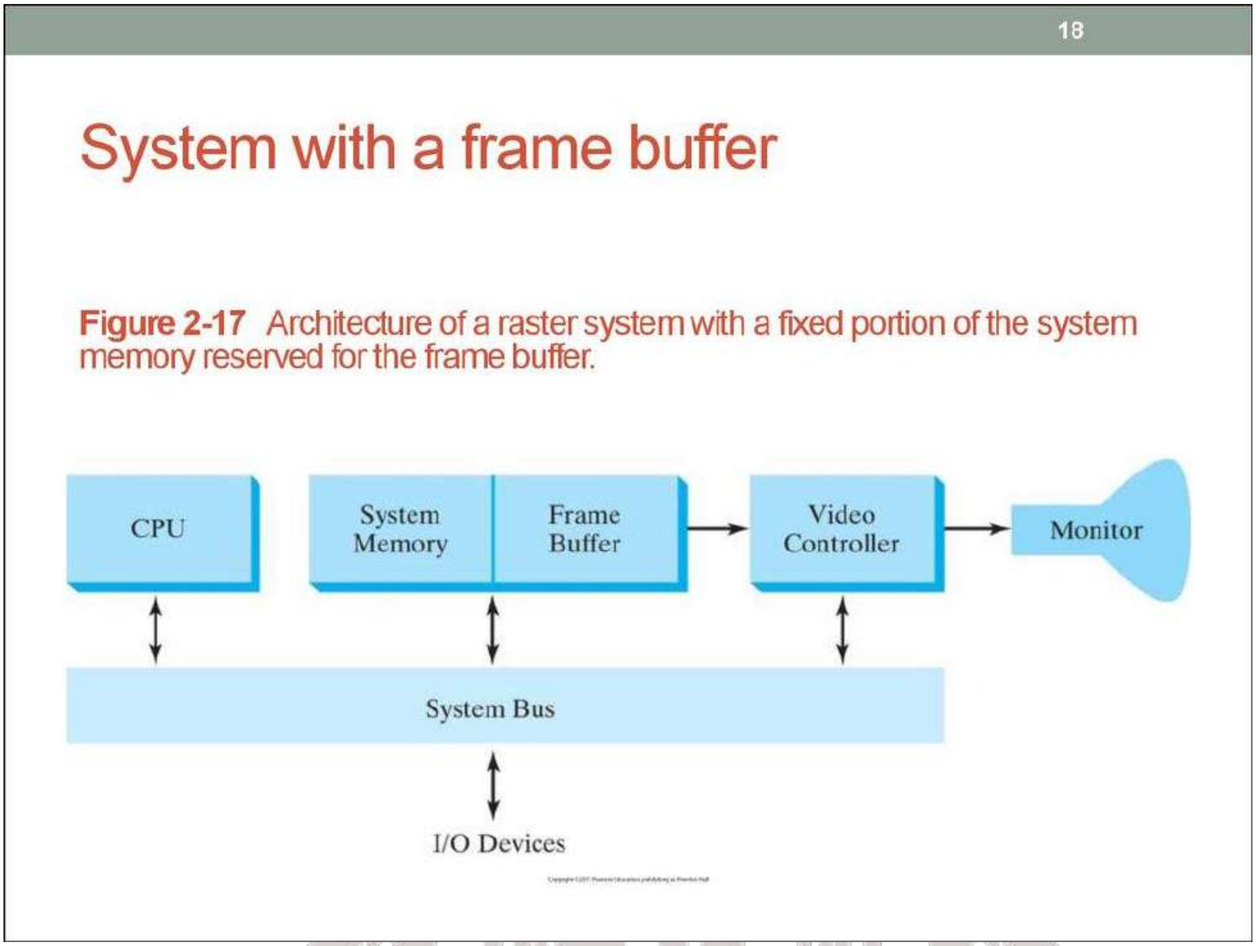


FIGURE: Architecture of a simple raster-graphics system.

Here the frame buffer can be anywhere in the system memory, and the video controller accesses the frame buffer to refresh the screen. In addition to the video controller, more sophisticated raster systems employ other processors as coprocessors and accelerators to implement various graphic operations.

3.1 Video Controller

The figure below shows a commonly used organization for raster systems. A fixed area of the system memory is reserved for the frame buffer, and the video controller is given direct access to the frame-buffer memory. Frame-buffer locations, and the corresponding screen positions, are referenced in the Cartesian coordinates.



System with a frame buffer

Figure 2-17 Architecture of a raster system with a fixed portion of the system memory reserved for the frame buffer.

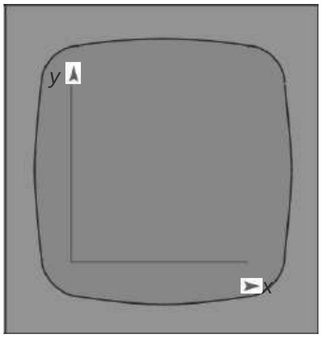
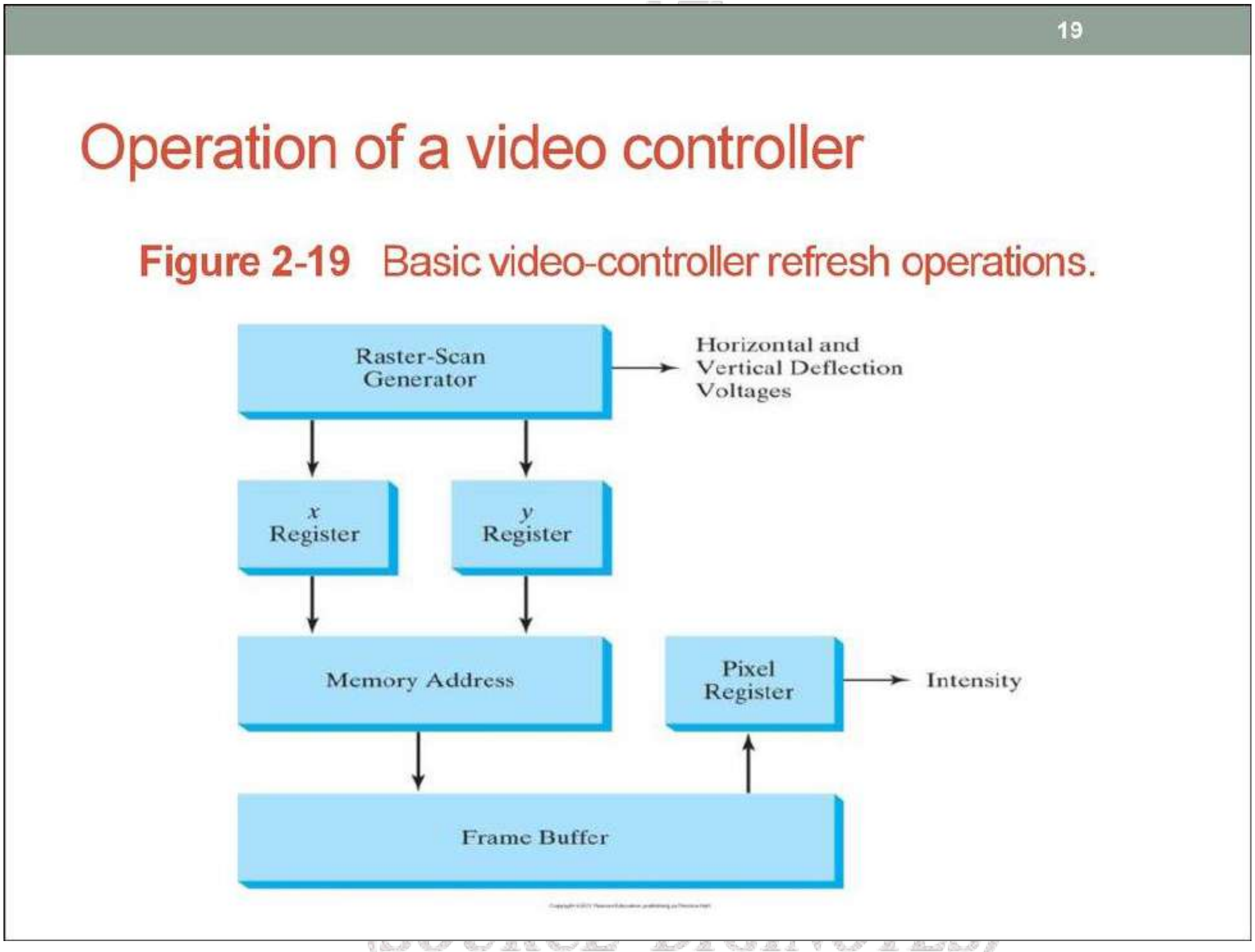


FIGURE 18

A Cartesian reference frame with origin at the lower-left corner of a video monitor.

The figure above shows a two-dimensional Cartesian reference frame with the origin at the lower-left screen corner. The screen surface is then represented as the first quadrant of a two-dimensional system, with positive X values increasing from left to right and positive Y values increasing from the bottom screen to top



19

Operation of a video controller

Figure 2-19 Basic video-controller refresh operations.

In the figure above, the basic refresh operations of the video controller are diagrammed. Two registers are used to store the coordinate values for the screen pixels. Initially, the X register is set to 0 and the Y register is set to the value for the top scan line. The contents of the frame buffer at this pixel position are then retrieved and used to set the intensity of the CRT beam.

A video controller can be designed to perform a number of other operations. For various applications, the video controller can retrieve pixel values from different memory areas on different refresh cycles.

3.2 Raster-Scan Display Processor:

The following figure shows one way to organize the components of a raster system that contains a separate display processor, sometimes referred to as a graphics controller or a display coprocessor. The purpose of the display processor is to free the CPU from the graphics chores. In addition to the system memory, a separate display-processor memory area can be provided.

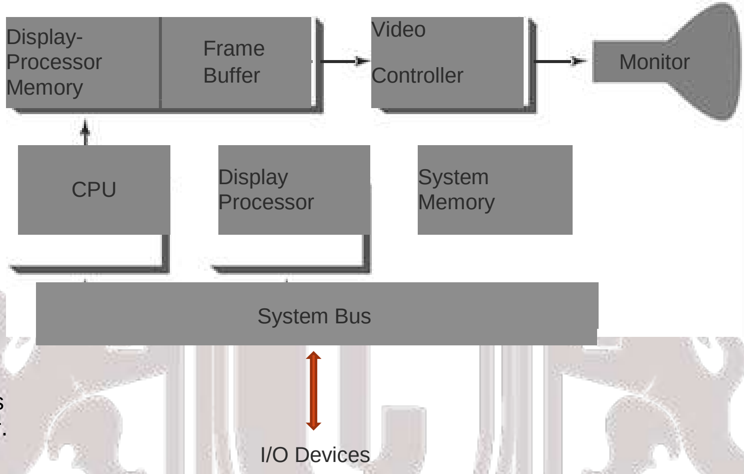


FIGURE 20
Architecture of a raster-graphics system with a display processor.

A major task of the display processor is digitizing a picture definition given in an application program into a set of pixel values for storage in the frame buffer. This digitization process is called Scan Conversion. Graphics commands specifying straight lines and other geometric objects are scan converted into a set of discrete points, corresponding to screen pixel positions.

Display processors are also designed to perform a number of additional operations. These Function include generating various line styles (dashed, dotted or solid), displaying color areas, and applying transformations to the objects in a scene. Also, display processors are typically designed to interface with interactive input devices, such as a mouse.

In an effort to reduce memory requirements in raster systems, methods have been devised for organizing the frame buffer as a linked list and encoding the color information. One organization scheme is to store each scan line as a set of numbered pairs.

Run-length encoding is a technique in which the first number in each pair can be a reference to a color value, and the second number specifies the number of adjacent pixels on the scan line that are to be displayed in that color. This technique result in a considerable saving in storage space if a picture is to be constructed mostly with long runs of a single color each.

Cell encoding is the approach of encoding the raster as a set of rectangular areas

Disadvantage of encoding runs:

- Color changes are difficult to record
- Storage requirements increases as the lengths of run decreases
- Difficult for the display controller to process raster when short runs are involved

Advantages of encoding methods:

Useful in digital storage and transmission of picture information

4. Graphics workstations and viewing systems

- Most graphics monitors today operate as raster-scan displays, and both CRT and flat panel systems are in common use.
- Graphics workstation range from small general-purpose computer systems to multi monitor facilities, often with ultra –large viewing screens.
- High-definition graphics systems, with resolutions up to 2560 by 2048, are commonly used in medical imaging, air-traffic control, simulation, and CAD.

Fig 1.1-A high-resolution (2048 by 2048) graphics monitor.

- Many high-end graphics workstations also include large viewing screens, often with specialized features.


Large-screen stereoscopic view

Fig 1.2- A large screen stereoscopic view of pressure contours in a vascular blood-flow simulation.

- Multi-panel display screens are used in a variety of applications that require "wall-sized" viewing areas. These systems are designed for presenting graphics displays at meetings, conferences, conventions, trade shows, retail stores etc.
- A multi-panel display can be used to show a large view of a single scene or several individual images. Each panel in the system displays one section of the overall picture.

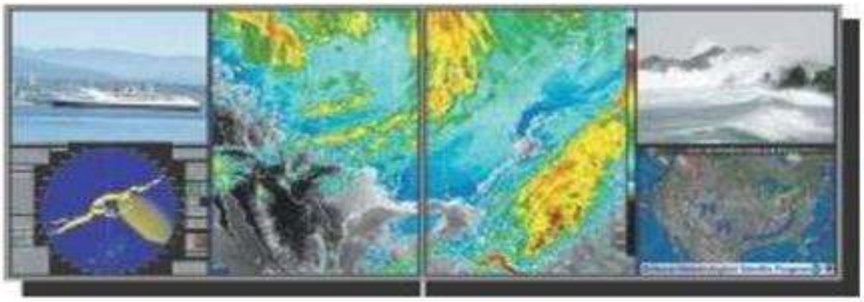

Multi-panel display

Fig 1.3- A multi-panel display system called the "Super Wall".

- Large graphics displays can also be presented on curved viewing screens.

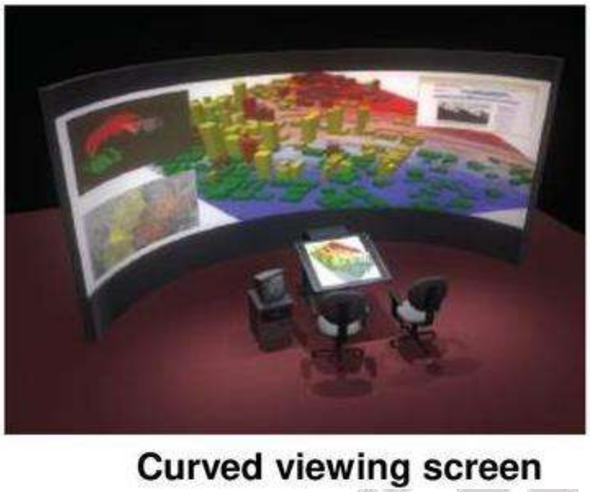


Fig 1.4- A homeland security study displayed using a system with large curved viewing screen.

- A large, curved-screen system can be useful for viewing by a group of people studying a particular graphics application.

Fig 1.5-A geophysical visualization presented on a 25 foot semicircular screen, which provides a 160 degree horizontal and 40 degree vertical field of view.

- A control center, featuring a battery of standard monitors, allows an operator to view sections of the large display and to control the audio, video, lighting, and projection systems using a touch-screen menu.
- A 360 degree paneled viewing system in the NASA control-tower simulator, which is used for training and for testing ways to solve air-traffic and runway problems at airports.

Fig 1.6-The 360 degree viewing screen in the NASA airport control tower simulator called the Future Flight Central Facility.

5. Input Devices

Graphics workstations make use of various devices for data input.Most systems have keyboards and mouses,while some other systems have trackball,spaceball,joystick,button boxes,touch panels,image scanners and voice systems.

Keyboard:

- Keyboard on graphics system is used for entering text strings,issuing certain commands and selecting menu options.
- Keyboards can also be provided with features for entry of screen coordinates,menu selections or graphics functions.
- General purpose keyboard uses function keys and cursor-control keys.
- Function keys allow user to select frequently accessed operations with a single keystroke.Cursor-control keys are used for selecting a displayed object or a location by positioning the screen cursor.

Button Boxes and Dials:

- Buttons are often used to input predefined functions .Dials are common devices for entering scalar values.

- Numerical values within some defined range are selected for input with dial rotations.

Mouse Devices:

- Mouse is a hand-held device,usually moved around on a flat surface to position the screen cursor.wheeler or roolers on the bottom of the mouse used to record the amount and direction of movement.

- Some of the mouses uses optical sensors,which detects movement across the horizontal and vertical grid lines.

- Since a mouse can be picked up and put down,it is used for making relative changes in the position of the screen.

- Most general purpose graphics systems now include a mouse and a keyboard as the primary input devices.

Trackballs and Spaceballs:

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

- Laptop keyboards are equipped with a trackball to eliminate the extra space required by a mouse.

- Spaceball is an extension of two-dimensional trackball concept.

- Spaceballs are used for three-dimensional positioning and selection operations in virtual-reality systems,modeling,animation,CAD and other applications.

Joysticks:

- Joystick is used as a positioning device,which uses a small vertical lever(stick) mounded on a base.It is used to steer the screen cursor around and select screen position with the stick movement.

- A push or pull on the stick is measured with strain gauges and converted to movement of the screen cursor in the direction of the applied pressure.

Data Gloves:

- Data glove can be used to grasp a virtual object.The glove is constructed with a series of sensors that detect hand and finger motions.

- Input from the glove is used to position or manipulate objects in a virtual scene.

Digitizers:

- Digitizer is a common device for drawing,painting or selecting positions.

- Graphics tablet is one type of digitizer,which is used to input 2-dimensional coordinates by activating a hand cursor or stylus at selected positions on a flat surface.

- A hand cursor contains cross hairs for sighting positions and stylus is a pencil-shaped device that is pointed at positions on the tablet.

Image Scanners:

- Drawings,graphs,photographs or text can be stored for computer processing with an image scanner by passing an optical scanning mechanism over the information to be stored.

- Once we have the representation of the picture, then we can apply various image-processing method to modify the representation of the picture and various editing operations can be performed on the stored documents.

Touch Panels:

- Touch panels allow displayed objects or screen positions to be selected with the touch of a finger.
- Touch panel is used for the selection of processing options that are represented as a menu of graphical icons.
- Optical touch panel-uses LEDs along one vertical and horizontal edge of the frame.
- Acoustical touch panels generates high-frequency sound waves in horizontal and vertical directions across a glass plate.

Light Pens:

- Light pens are pencil-shaped devices used to select positions by detecting the light coming from points on the CRT screen.
- To select positions in any screen area with a light pen,we must have some nonzero light intensity emitted from each pixel within that area.
- Light pens sometimes give false readings due to background lighting in a room.

Voice Systems:

- Speech recognizers are used with some graphics workstations as input devices for voice commands.The voice system input can be used to initiate operations or to enter data.
- A dictionary is set up by speaking command words several times,then the system analyses each word and matches with the voice command to match the pattern.

7. GRAPHICS NETWORKS:

So far, we have mainly considered graphics applications on an isolated system with a single user.

Multiuser environments & computer networks are now common elements in many graphics applications.

Various resources, such as processors, printers, plotters and data files can be distributed on a network & shared by multiple users.

A graphics monitor on a network is generally referred to as a graphics **server.**

The computer on a network that is executing a graphics application is called the **client.**

A workstation that includes processors, as well as a monitor and input devices can function as both a server and a client.

8. GRAPHICS ON INTERNET

A great deal of graphics development is now done on the **Internet**.
Computers on the Internet communicate using **TCP/IP**.
Resources such as graphics files are identified by **URL** (Uniform resource locator).

The **World Wide Web** provides a hypertext system that allows users to loacate and view documents, audio and graphics.

Each URL sometimes also called as universal resource locator.
The **URL** contains two parts

- **Protocol**- for transferring the document, and
- **Server**- contains the document.

9. Graphics Software

There are two broad classifications for computer-graphics software

➢ Special-purpose packages: Special-purpose packages are designed for nonprogrammers Example: generate pictures, graphs, charts, painting programs or CAD systems in some application area without worrying about the graphics procedures

➢ General programming packages: general programming package provides a library of graphics functions that can be used in a programming language such as C, C++, Java, or FORTRAN.

Example: GL (Graphics Library), OpenGL, VRML (Virtual-Reality Modeling Language), Java 2D, And Java 3D

NOTE: A set of graphics functions is often called a computer-graphics application programming interface (CG API)

Coordinate Representations

- To generate a picture using a programming package we first need to give the geometric descriptions of the objects that are to be displayed known as coordinates.
- If coordinate values for a picture are given in some other reference frame (spherical, hyperbolic, etc.), they must be converted to Cartesian coordinates.

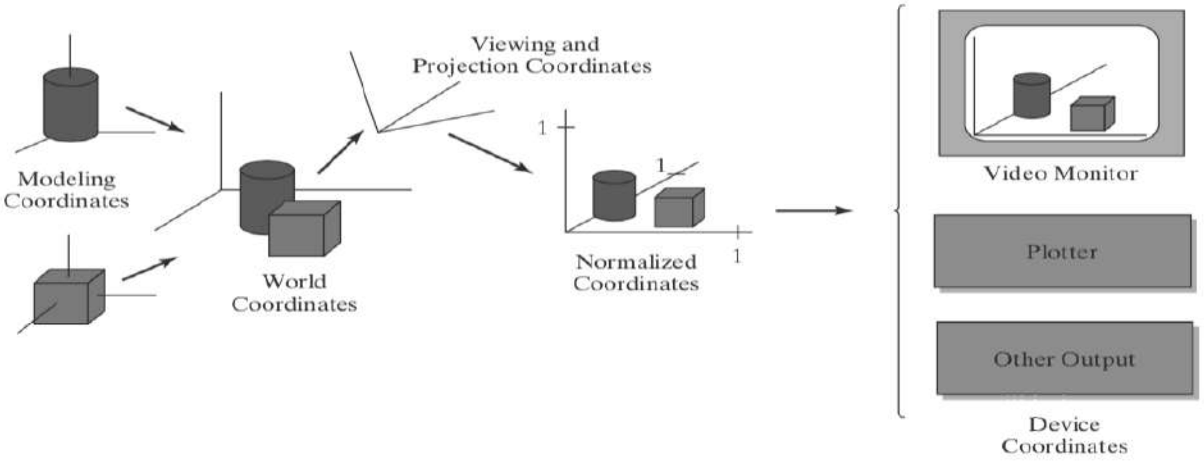
Several different Cartesian reference frames are used in the process of constructing and displaying

- First we define the shapes of individual objects, such as trees or furniture, These reference frames are called **modeling coordinates** or **local coordinates**
- Then we place the objects into appropriate locations within a scene reference frame called **world coordinates**.
- After all parts of a scene have been specified, it is processed through various output-device reference frames for display. This process is called the **viewing pipeline.**
- The scene is then stored in **normalized coordinates.** Which range from −1 to 1 or from 0 to 1 Normalized coordinates are also referred to as **normalized device coordinates**.
- .The coordinate systems for display devices are generally called **device coordinates**, or **screen coordinates.**

NOTE: Geometric descriptions in modeling coordinates and world coordinates can be given in floating-point or integer values.

Example: Figure briefly illustrates the sequence of coordinate transformations from modeling coordinates to device coordinates for a display

(xmc, ymc, zmc) → (xwc, ywc, zwc) → (xvc, yvc, zvc) → (xpc, ypc, zpc) → (xnc, ync, znc) → (xdc, ydc)



Computer Graphics Software

Graphics Functions

- It provides users with a variety of functions for creating and manipulating pictures
- The basic building blocks for pictures are referred to as **graphics output primitives**
- **Attributes** are properties of the output primitives
- We can change the size, position, or orientation of an object using **geometric transformations**
- **Modeling transformations**, which are used to construct a scene.
- **Viewing transformations** are used to select a view of the scene, the type of projection to be used and the location where the view is to be displayed.
- **Input functions** are used to control and process the data flow from these interactive devices(mouse, tablet and joystick)
- Graphics package contains a number of tasks .We can lump the functions for carrying out many tasks by under the heading **control operations.**

Software Standards

- The primary goal of standardized graphics software is **portability.**
- In 1984, **Graphical Kernel System (GKS)** was adopted as the first graphics software standard by the International Standards Organization (ISO)
- The second software standard to be developed and approved by the standards organizations was **Programmer's Hierarchical Interactive Graphics System (PHIGS).**

- Extension of PHIGS, called PHIGS+, was developed to provide **3-D** surface rendering capabilities not available in PHIGS.
- The graphics workstations from **Silicon Graphics, Inc. (SGI)**, came with a set of routines called **GL (Graphics Library)**

Other Graphics Packages

Many other computer-graphics programming libraries have been developed for

- general graphics routines
- Some are aimed at specific applications (animation, virtual reality, etc.)

Example: **Open Inventor**
Virtual-Reality Modeling Language (VRML)
We can create 2-D scenes with in Java applets (java2D, Java 3D)

10. Introduction To OpenGL:

OpenGL basic(core) library :-A basic library of functions is provided in OpenGL for specifying graphics primitives, attributes, geometric transformations, viewing transformations, and many other operations.

Basic OpenGL Syntax

-Function names in the **OpenGL basic library** (also called the **OpenGL core library**) are prefixed with **gl.** The component word first letter is capitalized.

For eg:- **glBegin, glClear, glCopyPixels, glPolygonMode**

-Symbolic constants that are used with certain functions as parameters are all in capital letters, preceded by "GL", and component are separated by underscore. For eg:-

GL_2D, GL_RGB, GL_CCW, GL_POLYGON, GL_AMBIENT_AND_DIFFUSE

-The OpenGL functions also expect specific data types. For example, an OpenGL function parameter might expect a value that is specified as a 32-bit integer. But the size of an integer specification can be different on different machines.

To indicate a specific data type, OpenGL uses special built-in, data-type names, such as **GLbyte, GLshort, GLint, GLfloat, GLdouble, GLboolean**

Related Libraries

-In addition to OpenGL basic(core) library(prefixed with gl), there are a number of associated libraries for handling special operations:-

1) OpenGL Utility(GLU):- Prefixed with “glu”. It provides routines for setting up viewing and projection matrices, describing complex objects with line and polygon approximations, displaying quadrics and B-splines using linear approximations, processing the surface-rendering operations, and other complex tasks.

-Every OpenGL implementation includes the GLU library

2) Open Inventor:- provides routines and predefined object shapes for interactive three-dimensional applications which are written in C++.

3) Window-system libraries:- To create graphics we need display window. We cannot create the display window directly with the basic OpenGL functions since it contains only device-independent graphics functions, and window-management operations are device-dependent. However, there are several window-system libraries that supports OpenGL functions for a variety of machines.

Eg:- Apple GL(AGL), Windows-to-OpenGL(WGL), Presentation Manager to OpenGL(PGL), GLX.

4) OpenGL Utility Toolkit(GLUT):- provides a library of functions which acts as interface for interacting with any device specific screen-windowing system, thus making our program device-independent. The GLUT library functions are prefixed with “**glut**”.

Header Files

In all graphics programs, we will need to include the header file for the OpenGL core library.

-In windows to include OpenGL core libraries and GLU we can use the following header files:-

```
#include <windows.h> //precedes other header files for including Microsoft windows ver
ofOpenGLlibraries
#include<GL/gl.h>
#include <GL/glu.h>
```

The above lines can be replaced by using GLUT header file which ensures gl.h and glu.h are included correctly,

```
#include <GL/glut.h>  //GL in windows
```

In Apple OS X systems, the header file inclusion statement will be,

```
#include <GLUT/glut.h>
```

Display-Window Management Using GLUT

Steps for displaying a picture:-

1) Initialization of GLUT:- the initialization function can also process command line arguments.

glutInit (&argc, argv);

2) Create a display window:-

glutCreateWindow ("An Example OpenGL Program");

The above function accepts a string which will be the title of display-window.

3) Specify content of display window:- For this, we create a picture using OpenGL functions and pass the picture definition to the GLUT routine glutDisplayFunc, which assigns our picture to the display window.

glutDisplayFunc (lineSegment); //passes the line-segment description to the display window.

4) Activate the display window:- the following line activates all the display windows, including their graphic content:

glutMainLoop ( );

This function must be the last one in our program. It displays the initial graphics and puts the program into an infinite loop that checks for input from devices such as a mouse or keyboard.

Additional GLUT functions:-

-We can specify location for the window using glutInitWindowPosition function

glutInitWindowPosition (50, 100);

The above statement specifies location that is 50 pixels to the right of the left edge of screen and 100 pixels down from top edge. The origin is at upper-left corner of the screen.

-Size of the display window can be specified using:

glutInitWindowSize (400, 300); //width of 400 pixels and height of 300 pixels

- We can also set a number of other options for the display window, such as buffering and a choice of color modes, with the **glutInitDisplayMode** function. The argument is a GLUT constant.

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // logical or('|') used to combine constants

The above statement specifies a single refresh buffer to be used and to use red, green, and blue components to select color values.

-Background color of window can be set using:

glClearColor (1.0, 1.0, 1.0, 0.0);

This statement set the background color to white. The first three parameter are for color where 1.0 is white and 0.0 is black. The fourth is called alpha value where 1.0 indicates opaque object and 0.0 indicates transparent object.

-To display the assigned color we use:

glClear (GL_COLOR_BUFFER_BIT);

The argument **GL_COLOR_BUFFER_BIT** is an OpenGL symbolic constant specifying that it is the bit values in the color buffer (refresh buffer) that are to be set to the values indicated in the **glClearColor** function.

-We can choose a variety of color schemes for the objects we want to display in a scene.

glColor3f (0.0, 0.4, 0.2);

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

-We need to tell OpenGL how we want to "project" our picture onto the display window because generating a two-dimensional picture is treated by OpenGL as a special case of three-dimensional viewing.

glMatrixMode (GL_PROJECTION);
gluOrtho2D (0.0, 200.0, 0.0, 150.0);

This specifies that an orthogonal projection is to be used to map the contents of a two-dimensional rectangular area of world coordinates to the screen, and that the *x*-coordinate values within this rectangle range from 0.0 to 200.0 with *y*-coordinate values ranging from 0.0 to 150.0. Whatever objects we define within this world-coordinate rectangle will be shown within the display window. Anything outside this coordinate range will not be displayed.

Therefore, the GLU function **gluOrtho2D** defines the coordinate reference frame within the display window to be (0.0, 0.0) at the lower-left corner of the display window and (200.0, 150.0) at the upper-right window corner.

The orthogonal projection just pastes our picture onto the screen.

The following code defines a two-dimensional, straight-line segment with integer, Cartesian endpoint coordinates (180, 15) and (10, 145).
glBegin(GL_LINES);
glVertex2i(180,15);
glVertex2i(10,145);
glEnd ( );

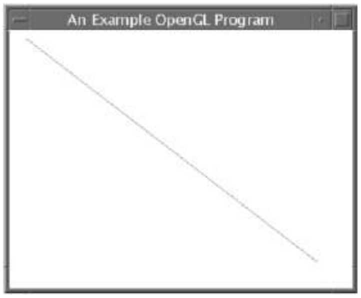
-glFlush:- routine to force execution of our OpenGL functions, which are stored by computer systems in buffers in different locations

Q) Program to display a two-dimensional line segment.

```
#include<GL/glut.h> // (or others, depending on the system in use)
void init (void)
{
glClearColor(1.0, 1.0, 1.0, 0.0); // Set display-window color to white.
glMatrixMode(GL_PROJECTION); // Set projection parameters.
gluOrtho2D(0.0,200.0,0.0,150.0);
}
voidlineSegment(void)
{
glClear(GL_COLOR_BUFFER_BIT);//Clear display window.
glColor3f(0.0,0.4,0.2);// Set line segment color to green.
glBegin(GL_LINES);
glVertex2i (180, 15); // Specify line-segment geometry.
glVertex2i (10, 145);
glEnd ( );
glFlush ( ); // Process all OpenGL routines as quickly as possible.
}
void main (int argc, char** argv)
{
glutInit (&argc, argv); // Initialize GLUT.
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // Set display mode.
glutInitWindowPosition (50, 100); // Set top-left display-window position.
glutInitWindowSize (400, 300); // Set display-window width and height.
glutCreateWindow ("An Example OpenGL Program"); // Create display window.
init ( ); // Execute initialization procedure.
glutDisplayFunc (lineSegment); // Send graphics to display window.
glutMainLoop ( ); // Display everything and wait.

}
```

OUTPUT:-



-The procedure **lineSegment** is referred to as a *display callback function.* And this procedure is described as being “registered” by **glutDisplayFunc** as the routine to invoke whenever the display window might need to be redisplayed. This can occur, for example, if the display window is moved.

11. Coordinate Reference Frames.

To describe a picture, we first decide upon a convenient Cartesian coordinate system, called the **world-coordinate reference frame** which could be either Two-dimensional or Three-dimensional.

Coordinate positions are stored along with other information about the objects, such as their color and their **coordinate extents.**
Coordinate extents are the minimum and maximum x, y and z values for each object.
A set of Coordinate Extents is also described as a **bounding box** for an object

The scan-conversion process stores information about the scene, such as color values, at the appropriate locations in the frame buffer, and the objects in the scene are displayed on the output device.

Screen Coordinates

Locations on a video monitor are referenced in integer screen coordinates, which correspond to the pixel positions in the frame buffer.

Pixel Coordinate values give the scan line number and the column number.
Scan lines are referenced from 0, at the top of the screen, to some integer value, y_{max} at the bottom of the screen and pixel positions along each scan line are numbered from 0 to x_{max} , left to right.

A display algorithm must calculate the positions for those pixels that lie along the line path between the endpoints. A pixel position occupies a finite area of the screen,

Once a pixel positions have been identified for an object, the appropriate color values must be stored in the frame buffer.

setPixel (x, y) ;

This procedure stores the current color setting into the frame buffer at integer position (x, y) relative to the selected position of the screen-coordinate origin.

getPixel (x, y, color) ;

Screen coordinates are stored as three-dimensional values, where the third dimension references the depth of object positions relative to a viewing position. For a two-dimensional scene, all depth values are 0.

Absolute and Relative Coordinate Specifications

Absolute coordinate values means that the values specified are the actual positions within the coordinate system in use.

Some graphics packages also allow positions to be specified using relative coordinates. This method is useful for various graphics applications, such as producing drawings with pen plotters, artist's drawing and painting systems, and graphics packages for publishing and printing applications.

We can specify a coordinate position as an offset from the last position that was referenced called as the **current position.**

12. Specifying a two-dimensional world-coordinate reference frame in OpenGL:

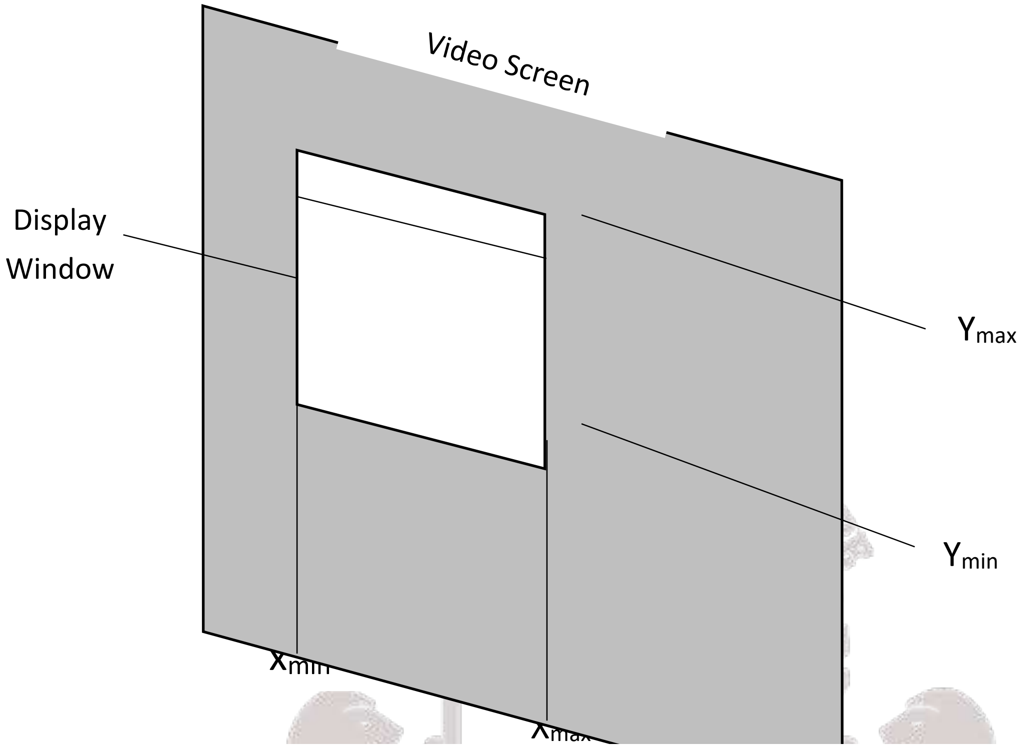
The **gluOrtho2D** command is a function used to set up any two-dimensional Cartesian reference frame. The arguments for this function are the four values defining the x and y coordinate limits for the picture to display.

The **gluOrtho2D** function specifies an orthogonal projection and the coordinate values are placed in the OpenGL projection matrix and then assign the identity matrix as the projection matrix before defining the old coordinate range.

It ensures that the coordinate values are not accumulated with any values which have been set for previous projection matrix.

glMatrixMode (GL_PROJECTION) ;
glLoadIdentity ( ) ;
gluOrtho2D (x_{min}, x_{max}, y_{min}, y_{max}) ;

The display window will then be referenced by coordinates (x_{min}, y_{min}) at the lower left-corner and by coordinates (x_{max}, y_{max}) at the upper-right corner as shown below.



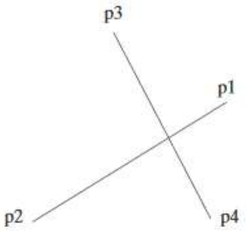
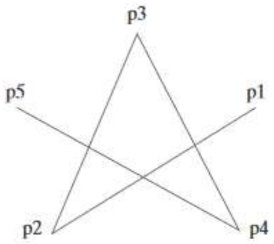
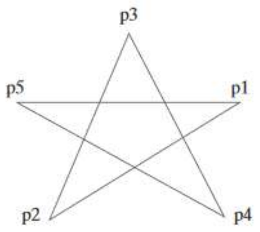
World coordinate limits for a display window, as specified in the gluOrtho2D

If the coordinate extents are within the coordinate range of display window, all primitives will be displayed otherwise, only parts of the primitives within the display window and all positions for the OpenGL primitives must be given in absolute coordinates with respect to the reference frame defined in the gluOrtho2D function.

13. OpenGL Line Functions

- Graphics packages typically provide a function for specifying one or more straight-line segments, where each line segment is defined by two endpoints coordinate positions.
- We use a symbolic constant as the argument for the glBegin function that interprets a list of positions as the endpoint coordinates for line segments.
- There are three symbolic constants in OpenGL that we can use to specify how a list of endpoint positions should be connected to form a set of straight-line segments.

GL_LINES	GL_LINE_STRIP	GL_LINE_LOOP
A set of straight-line segments between each	The display is a sequence of connected line segments	An additional line is

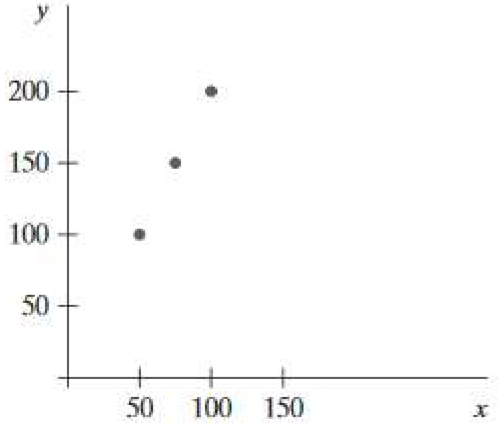
successive pair of endpoints in a list is generated using this primitive line constant.	between the first endpoint in the list and the last endpoint which is a **polyline**.	drawn to connect the last coordinate position and the first coordinate position which produces a **closed polyline**.
A set of unconnected lines are formed unless some coordinate positions are repeated. Lines that cross but do not share a vertex are still considered to be unconnected.	The first line segment in the polyline is displayed between the first endpoint and the second endpoint; the second line segment is between the second and third endpoints; and so on, up to the last line endpoint.	The first line segment in the polyline is displayed between the first endpoint and the second endpoint; the second line segment is between the second and third endpoints; and so on, up to the last line endpoint which in turn is connected to the first endpoint.
Nothing is displayed if only one endpoint is specified, and the last endpoint is not processed if the number of endpoints listed is odd.	Nothing is displayed if at least two coordinate positions are not listed.	Nothing is displayed if at least two coordinate positions are not listed.
Code to display the following figure:  glBegin(GL_LINES); glVertex2iv (p1); glVertex2iv (p2); glVertex2iv (p3); glVertex2iv (p4); glVertex2iv (p5); glEnd ();	Code to display the following figure:  glBegin(GL_LINE_STRIP); glVertex2iv (p1); glVertex2iv (p2); glVertex2iv (p3); glVertex2iv (p4); glVertex2iv (p5); glEnd ();	Code to display the following figure:  glBegin(GL_LINE_LOOP); glVertex2iv (p1); glVertex2iv (p2); glVertex2iv (p3); glVertex2iv (p4); glVertex2iv (p5); glEnd ();

14. OpenGL Point Functions

- A coordinate position in the world reference frame is given to specify the geometry of a point. Then this coordinate position, along with other geometric descriptions we may have in our scene, is passed to the viewing routines.
- OpenGL primitives are displayed with a default size and colour if other attribute values are not specified.
- We use the following OpenGL function to state the coordinate values for a single position:
  glVertex* ();
  The asterisk (*) indicates that suffix codes are required for this function.
- These suffix codes are used to identify the spatial dimension, the numerical data type to be used for the coordinate values, and a possible vector form for the coordinate specification.
- Calls to **glVertex** functions must be placed between a **glBegin** function and a **glEnd** function. The argument of the **glBegin** function is used to identify the kind of output primitive that is to be displayed, and **glEnd** takes no arguments.
- For point plotting, the argument of the **glBegin** function is the symbolic constant **GL_POINTS**.

```
glBegin (GL_POINTS);
glVertex* ();
glEnd ();
```

- The **glVertex** function is used in OpenGL to specify coordinates for any point position.
- Coordinate positions in OpenGL can be given in two, three, or four dimensions. We use a suffix value of 2, 3, or 4 on the **glVertex** function to indicate the dimensionality of a coordinate position.
- The second suffix code on the **glVertex** function is used to specify the numerical values of the coordinates.
- Suffix codes for specifying a numerical data type are i(integer), s(short), f(float), and d(double).



Display of three-point positions generated with **glBegin**(**GL_POINTS**).

- Coordinate values can be listed explicitly in the **glVertex** function:

```
glBegin (GL_POINTS);
glVertex2i (50, 100);
glVertex2i (75, 150);
glVertex2i (100, 200);
glEnd ();
```

- Or a single argument can be used that references a coordinate position as an array. If we use an array specification for a coordinate position, we need to append v("vector") as a third suffix code:

```
int point1 [ ] = {50, 100};
int point2 [ ] = {75, 150};
int point3 [ ] = {100, 200};
```

- Call the OpenGL functions:

```
glBegin (GL_POINTS);
glVertex2iv (point1);
glVertex2iv (point2);
glVertex2iv (point3);
glEnd ();
```

15. Point Attributes

Basically, we can set two attributes for points: color and size.\

OpenGL Point-Attribute Functions

The displayed color of a designated point position is controlled by the current color values in the state list. Also, a color is specified with either the **glColor** function.

We set the size for an OpenGL point with

glPointSize (size);

and the point is then displayed as a square block of pixels. Parameter **size** is assigned a positive floating-point value, which is rounded to an integer.

16. Line Attributes:

A straight-line segment can be displayed with three basic attributes: **color, width,** and **style**.
Line color is typically set with the same function for all graphics primitives, while line width and line style are selected with separate line functions.

OpenGL Line-Attribute Functions:

We can control the appearance of a straight-line segment in OpenGL with three attribute settings: line color, line width, and line style.

We have already seen how to make a color selection, and OpenGL provides a function for setting the width of a line and another function for specifying a line style, such as a dashed or dotted line.

OpenGL Line-Width Function

Line width is set in OpenGL with the function

glLineWidth (width);

We assign a floating-point value to parameter **width**, and this value is rounded to the nearest nonnegative integer.

OpenGL Line-Style Function

By default, a straight-line segment is displayed as a solid line.

However, we can also display dashed lines, dotted lines, or a line with a combination of dashes and dots, and we can vary the length of the dashes and the spacing between dashes or dots.

We set a current display style for lines with the OpenGL function

glLineStipple (repeatFactor, pattern);

Parameter **pattern** is used to reference a 16-bit integer that describes how the line should be displayed.

A 1 bit in the pattern denotes an “on” pixel position, and a 0 bit indicates an “off” pixel position.

The pattern is applied to the pixels along the line path starting with the low-order bits in the pattern.

The default pattern is 0xFFFF (each bit position has a value of 1), which produces a solid line.

Integer parameter **repeatFactor** specifies how many times each bit in the pattern is to be repeated before the next bit in the pattern is applied. The default repeat value is 1.

17. Line Drawing Algorithm

A straight-line segment in a scene is defined by coordinate positions for the endpoints of the segment.

To display the line on a raster monitor, the graphics system must first project the endpoints to integer screen coordinates and determine the nearest pixel positions along the line path between the two endpoints then the line color is loaded into the frame buffer at the corresponding pixel coordinates.

A computed line positions of (10.48,20.51) is converted to pixel position (10,21). This rounding of coordinates values to integers causes all but horizontal and vertical lines to be displayed with a stair-step appearance ("the jaggies").

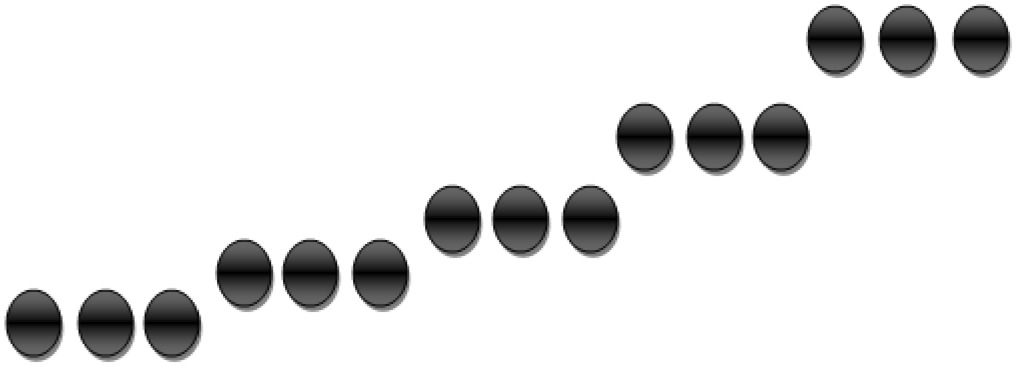


Fig: stair-step effect(jaggies) produced when a line is generated as a series of pixel positions.

17.1 Line Equations

The Cartesian slope-intercept equation for a straight line is

$$y=m * x +b\text{------------}>(1)$$

with m as the slope of the line and b as the y intercept.

Given that the two endpoints of a line segment are specified at positions (x_0,y_0) and (x_{end}, y_{end}) ,as shown in fig.

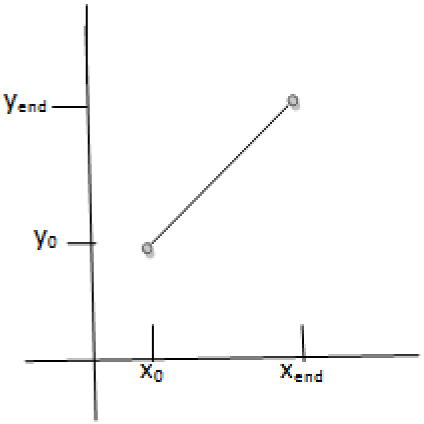


fig. Line path between endpoint positions (x_0,y_0) and (x_{end}, y_{end}).

we determine values for the slope m and y intercept b with the following equations:

$m=(y_{end} - y_0)/(x_{end} - x_0)$----------------->(2)

$b=y_0 - m.x_0$-------------->(3)

Algorithms for displaying straight line are based on the line equation (1) and calculations given in eq(2) and (3).

for given x interval δx along a line, we can compute the corresponding y interval δy from eq.(2) as

$\delta y=m. \delta x$----------------->(4)

similarly, we can obtain the x interval δx corresponding to a specified δy as

$\delta x=\delta y/m$------------------>(5)

These equations form the basis for determining deflection voltages in analog displays, such as vector-scan system, where arbitrarily small changes in deflection voltage are possible.

For lines with slope magnitudes

- |m|<1, δx can be set proportional to a small horizontal deflection voltage with the corresponding vertical deflection voltage set proportional to δy from eq.(4)
- |m|>1, δy can be set proportional to a small vertical deflection voltage with the corresponding horizontal deflection voltage set proportional to δx from eq.(5)
- |m|=1, δx=δy and the horizontal and vertical deflections voltages are equal

On raster systems, lines are plotted with pixels ,and the step sizes in the horizontal and vertical directions are constrained by pixel separations. That is we must "sample" a line at discrete positions and determine the nearest pixel to the line at each sample position. The scan-conversion process for straight lines is illustrated in fig. with discrete sample positions along the x -axis.

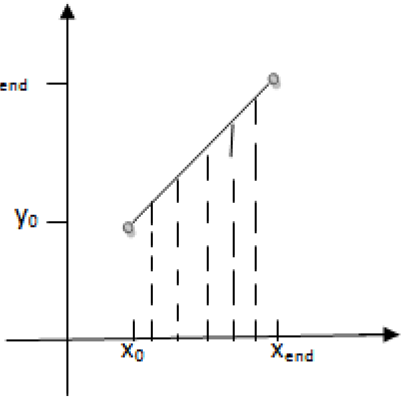


Fig. straight line segment with five sampling positions along x-axis between x_0 and x_{end}

17.2 DDA Algorithm (DIGITAL DIFFERENTIAL ANALYZER)

The DDA is a scan-conversion line algorithm based on calculating either δy or δx.

A line is sampled at unit intervals in one coordinate and the corresponding integer values nearest the line path are determined for the other coordinate

DDA Algorithm has three cases so from equation i.e.., $m=(y_{k+1} - y_k)/(x_{k+1} - x_k)$

Case1:if m<1,x increment in unit intervals

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

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

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

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

where k takes integer values starting from 0,for the first point and increases by 1 until final endpoint is reached. Since m can be any real number between 0.0 and 1.0, Case2:if m>1, y increment in unit intervals

i.e.., $y_{k+1} = y_k + 1$

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

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

$x_{k+1} =(1/m)+ x_k$-----------------(2)

Case3:if m=1,both x and y increment in unit intervals

i.e..,$x_{k+1}=x_k + 1$ and y= $y_{k+1} + 1$

Equations (1) and (2) are based on the assumption that lines are to be processed from the left endpoint to the right endpoint. If this processing is reversed, so that the starting endpoint is at the right, then either we have δx=-1 and

$y_{k+1} = y_k - m$-----------------(3)

or(when the slope is greater than 1)we have δy=-1 with

$x_{k+1} = x_k - (1/m)$----------------(4)

Similar calculations are carried out using equations (1) through (4) to determine the pixel positions along a line with negative slope. thus, if the absolute value of the slope is less than 1 and the starting endpoint is at left ,we set δx==1 and calculate y values with eq(1).

when starting endpoint is at the right(for the same slope),we set δx=-1 and obtain y positions using eq(3).

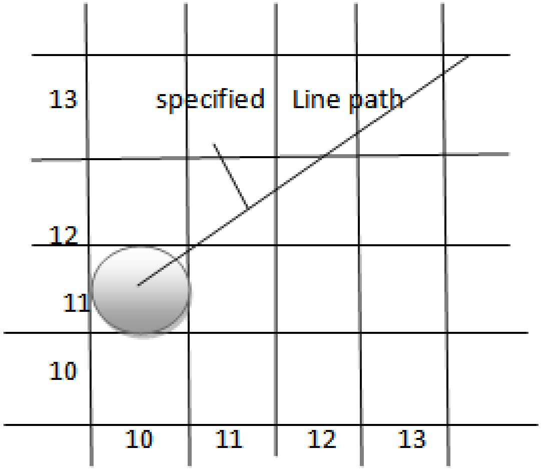


fig. A section of a display screen where a straight line segment is plotted

For negative slope with absolute value greater than 1,we use δy=-1 and eq (4) or we use δy=1 and eq(2).

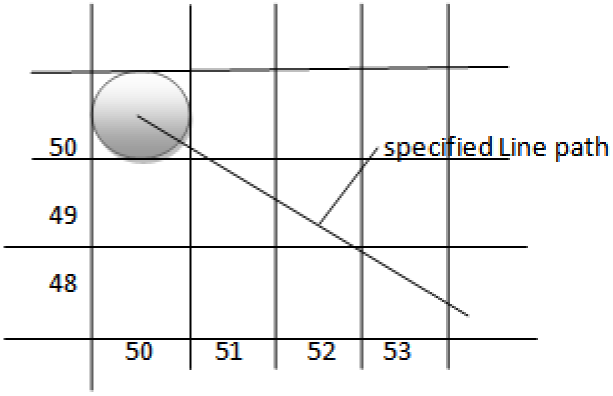


fig. A section of a display screen where a negative slope line segment is plotted

This algorithm is summarized in the following procedure, which accepts as input two integer screen positions for the endpoints of a line segment.

if m<1,where x is incrementing by 1

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

- So initially x=0,Assuming (x_0,y_0)as initial point assigning x= x_0,y=y_0 which is the starting point .

Illuminate pixel(x, round(y))

- x_1= x+ 1 , y_1=y + 1

  Illuminate pixel(x_1,round(y_1))

- x_2= x_1+ 1 , y_2=y_1 + 1

  Illuminate pixel(x_2,round(y_2))

.............Till it reaches final point.

if m>1,where y is incrementing by 1

x_{k+1} =(1/m)+ x_k

- So initially y=0,Assuming (x_0,y_0)as initial point assigning x= x_0,y=y_0 which is the starting point .

Illuminate pixel(round(x),y)

- x_1= x+(1/m) ,y_1=y

  Illuminate pixel(round(x_1),y_1)

- x_2= x_1+ (1/m) , y_2=y_1

  Illuminate pixel(round(x_2),y_2)

.............Till it reaches final point.

The DDA algorithm is faster method for calculating pixel position than one that directly implements .

It eliminates the multiplication by making use of raster characteristics, so that appropriate increments are applied in the x or y directions to step from one pixel position to another along the line path.

The accumulation of round off error in successive additions of the floating point increment, however can cause the calculated pixel positions to drift away from the true line path for long line segments. Furthermore ,the rounding operations and floating point arithmetic in this procedure are still time consuming.

we improve the performance of DDA algorithm by separating the increments m and 1/m into integer and fractional parts so that all calculations are reduced to integer operations.

EXAMPLE:

1) Consider two points (2,3) and (12,8) solve by using DDA Algorithm.

solution :: Assign $(x_1,y_1)=(2,3)$ and $(x_2,y_2)=(12,8)$

$m=(y_2 - y_1)/(x_2 - x_1) = (8-3)/(12-2) = 0.5$

It is in case1 :m<1

$(x_1,y_1)=(2,3)$

$x_2=x_1 + 1$ $y_2 = y_1 + m$

$x_2 = 2+1 = 3$ $y_2 = 3+0.5 = 3.5$

illuminate pixel(3,round(3.5)) = (3,3.5)

$(x_2,y_2)=(3,4)$

$x_3 = x_2 + 1$ $y_3 = y_2 + m$

$x_3 = 3+1 = 4$ $y_3 = 3.5+0.5 = 4$

illuminate pixel(4,round(4)) = (4,4)

$(x_3,y_3) = (4,4)$

$x_4 = x_3 + 1$ $y_4 = y_3 + m$

$x_4 = 4+1 = 5$ $y_4 = 4+0.5 =4.5$

X	Y	x-plot	y-plot
2	3	2	3
3	3.5	3	4
4	4	4	4
5	4.5	5	5
6	5	6	5

7	5.5	7	6
8	6	8	6
9	6.5	9	7
10	7	10	7
11	7.5	11	8
12	8	12	8

17.3 Bresenham's Algorithm:

It is an efficient raster scan generating algorithm that uses incremental integral calculations.

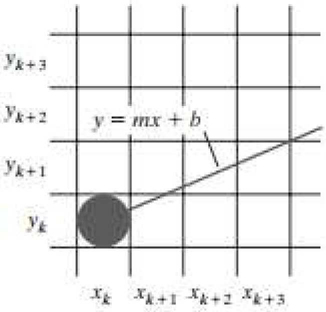


FIGURE 6
A section of the screen showing a pixel in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope $0 < m < 1$.

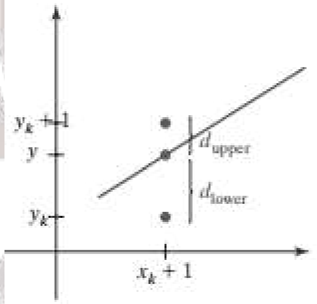


FIGURE 7

Assumptions:

1) Consider two points with coordinates (x_1, y_1) and (x_2, y_2) of a line.

2) The slope of the line i.e., m<=45

3) $x_1 < x_{2--}$

4) Consider the equation of a straight line y=mx+c where m=dy/dx

ALGORITHM:

- Here we have to decide upon which pixel the line has to be drawn.
- The first pixel (x_i, y_i) is selected without any confusion but in the next line is passed on the two pixels i.e., $(x_i, y_i)$and (x_{i+1}, y_{i+1}).Hence through this algorithm we need to select which pixel has to be selected.
- This decision has to be made by taking the distances d1 and d2.
- Since the equation of line is y=mx+c and next point is x_{i+1}

'y' can be calculated as,

$Y=m(x_i +1)+b$ --->(1)

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

Consider $d1=y-y_i$

Replace y with eqn(1),

$d1=m(x_i +1)+b-y_i$

Consider $d2=y_{i+1} -y$

$d2=y_i + 1-y$

replace eqn(1) to y,

$d2=y_i +1-m(x_i +1)-b$

Lets subtract,

d1-d2

$=>m(x_i +1)+b-y_i-y_i-1+m(x_i +1)+b$

$=>2m(x_i +1)+2b-2y_i-1$

$=>2[m(x_i +1)-2y_i+2b-1$--->(2)

We subtracted d1-d2,because to know which distance is near to the line.

- If (d1-d2)<0 means d1<d2 where d1 is nearer to the line[closer to y_i].Hence select the point(x_{i+1},y_i)
  $=>y_{i+1}=y_i$

- If (d1-d2)>0 means d1>d2 where d2 is nearer to the line[closer to y_{i+1}].Hence select the point (x_{i+1},y_{i+1})

  =>y_{i+1}=y_i + 1

As WKT,m=Δy/Δx which always gives fraction value and Bresenham's algorithm avoids this fraction values,so eqn(2) will be multiplied by Δx,

i.e,(d1-d2=2m(x_i + 1)-2y_i+2b-1)*Δx

Let us consider the new equation variable P_i

i.e.,P_i=(d1-d2)Δx

Here the sign of Pi=sign of (d1-d2)

If P_i<0 then d1-d2<0 and so on.

P_i=2.Δy.x_i+2.Δy-2.y_i.Δx+2.Δx.b-Δx---(3)

=2.Δy.x_i-2.y_i.Δx+(2.Δy+2.Δx.b-Δx)

From the above eqn (2.Δy+2.Δx.b-Δx) is considered constant.

C=2.Δy+2.Δx.b-Δx

Hence eqn becomes,

P_i=2.Δy.x_i-2.y_i.Δx+C---(4)

As the sign of P_i is same as (d1-d2),P_i can be used to decide which pixel needs to be selected.

Lets consider the next iteration for P_i i.e, P_{i+1} and eqn is

P_{i+1}=2.Δy.x_{i+1} - 2.y_{i+1}.Δx+C---(5)

Subtract P_{i+1}-P_i,

=2.Δy(x_{i+1}-x_i)-2.Δx(y_{i+1}-y_i)

x_{i+1} is nothing but moving to next pixel where each pixel is one unit difference.

So,x_{i+1} can also be written as

$x_{i+1}=x_i + 1$ which can be replaced in the above eqn.

$P_{i+1} - P_i=2.\Delta y-2.\Delta x(y_{i+1}-y_i)$---(6)

while y_{i+1} is not equal to $y_i + 1$,because the value of y will always not be incremented,it can either stay in y_i or y_{i+1}.

Move P_i to RHS,

$P_{i+1}=P_i+2.\Delta y-2.\Delta x(y_{i+1}-y_i)$

Simplify the above eqns as,

If $P_i<0$,

Then replace y_{i+1}->y_i

$P_{i+1}=P_i+2.\Delta y-2.\Delta x(y_i-y_i)$

$P_{i+1}=P_i+2.\Delta y$---(7)

If $P_i>0$,

Then replace y_{i+1}->$y_i +1$

$P_{i+1}=P_i+2.\Delta y-2.\Delta x$---(8)

Hence the equation for algorithm depends on eqn(7) and eqn(8),

$P_{i+1}=P_i+2.\Delta y$

$P_{i+1}=P_i+2.\Delta y-2.\Delta x$

Now we have to find the intial parameters by considering the eqn(3),

$P_i=2.\Delta y.x_i-2.\Delta x.y_i+2.\Delta y+2.\Delta x.b-\Delta x$

Assigning i=0,

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

Now from y=mx+b

$b=y_0-mx_0$

$b=y_0-(\Delta y/\Delta x).x_0$

Put $b=y_0-(\Delta y/\Delta x).x_0$ in the eqn above,

$P_0=2.\Delta y.x_0-2.\Delta x.y_0+2.\Delta y+2.\Delta.x[y_0- (\Delta y/\Delta x)x_0]-\Delta.x$

$P_0=2\Delta.y-\Delta.x$-----Initial Parameter

Bresenham's Line-Drawing Algorithm for |m| < 1.0

1. Input the two line endpoints and store the left endpoint in (x_0, y_0).

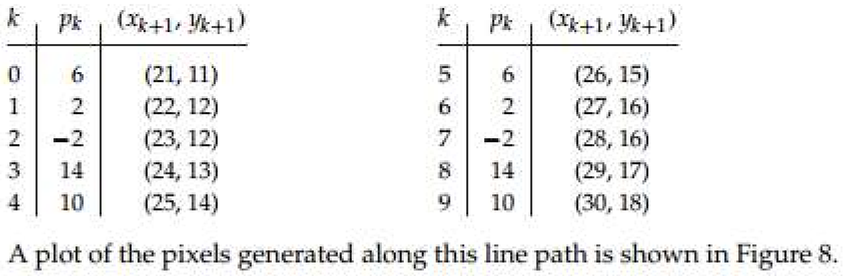
2. Set the color for frame-buffer position (x_0, y_0); i.e., plot the first point.

3. Calculate the constants x, y, 2y, and 2y − 2x, and obtain the starting value for the decision parameter as $p_0 = 2y - x$

4. At each xk along the line, starting at k = 0, perform the following test: If pk < 0, the next point to plot is (xk + 1, yk) and pk+1 = pk + 2y Otherwise, the next point to plot is (xk + 1, yk + 1) and pk+1 = pk + 2y − 2x

5. Repeat step 4 x − 1 more times.

EXAMPLE 1 Bresenham Line Drawing To illustrate the algorithm, we digitize the line with endpoints (20, 10) and (30, 18). This line has a slope of 0.8, with x = 10, y = 8 The initial decision parameter has the value $p_0 = 2y - x = 6$ and the increments for calculating successive decision parameters are 2y = 16, 2y − 2x = −4 We plot the initial point (x_0, y_0) = (20, 10), and determine successive pixel positions along the line path from the decision parameter as follows:



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

A plot of the pixels generated along this line path is shown in Figure 8.

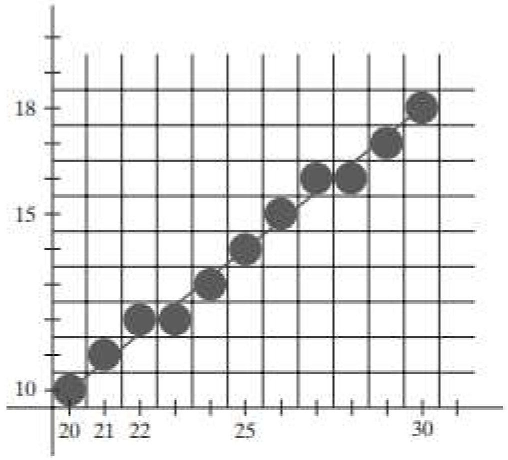


FIGURE 8
Pixel positions along the line path between endpoints (20, 10) and (30, 18), plotted with Bresenham's line algorithm.

17.4 Midpoint Circle Algorithm

- Midpoint circle algorithm generates all points on a circle centered at the origin by incrementing all the way around circle.
- The strategy is to select which of 2 pixels is closer to the circle by evaluating a function at the midpoint between the 2 pixels

Eight way symmetry

The shape of the circle is similar in each quadrant.Therefore ,if we determine the curve positions in the first quadrant ,we can generate the circle positions in the second quadrant of xy plane.The circle sections in the third and fourth quadrant can be obtained from sections in the first and second quadrant by considering the symmetry along X axis

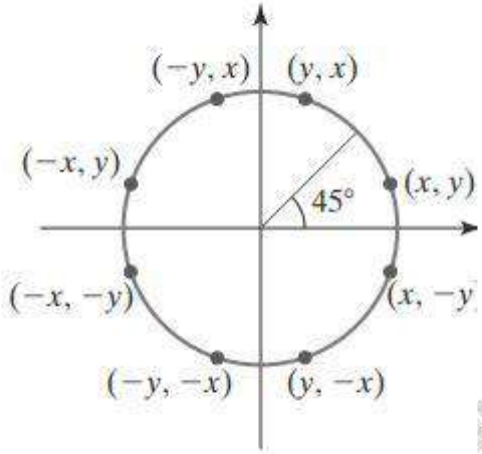
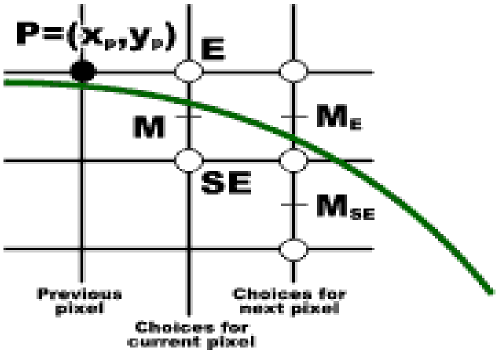


FIGURE 13

Symmetry of a circle. Calculation of a circle point (x, y) in one octant yields the circle points shown for the other seven octants.

  - Conside the circle centered at the origin,if the point (x, y) is on the circle,then we can compute 7 other points on the circle as shown in the above figure.
  - So we need to compute only one 45 degree segment to determine the circle completely.
- In **Midpoint Circle Algorithm** by evaluating a function at the midpoint between the 2 pixel,we can decide which pixel is closer to the circle.

- If pixel P at (x_p,y_p) has been chosen as the starting pixel,then the next pixel can be E or SE as shown in figure below.
- Now the choice is between E or SE



where E= (x_{p+1}, y_p)

SE= (x_{p+1} , y_{p-1})

M=(x_{p+1} , y_p - 1/2)

- According to implicit formula:

  F(x,y) = $x^2 + y^2 - R^2$, we use implicit formula for this algorithm.

- The relative position of any point (x ,y) can be determined by checking the sign of the circle function

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

- These tests are performed for the midpositions between pixels near the circle path at each sampling step.Thus the circle function is the decision parameter in the midpoint algorithm.
- If the midpoint **between** the pixel E and SE is outside the circle,then SE is closer to the circle
- If the midpoint is **inside** the circle,then pixel E is closer to the circle.
- As for lines,we choose on the basis of the decision variable d,which is the value of the function at the midpoint,

  p_k =F(x_p+1 , y_p - 1/2)

  = $(x_p + 1)^2 + (y_p - 1/2)^2 - R^2$

➢ If $p_k >= 0$

  - SE is chosen ,where x is incremented by 1 and y is decremented by 1.

    P_{k+1}=F(x_p + 2, y_p – 3/2)

    $\Delta d_{se} = p_{k+1} - p_k$

$= F(x_p+2, y_p-3/2) – F (x_p+1 ,y_p – ½)$

$=(x_p + 2)^2 + (y_p – 3/2)^2 – R^2 – \{(x_p +1)^2 +$
$(y_p – ½)^2 - R^2$

Expand this equation using the formula $(a + b)^2$ and $(a - b)^2$

$=\{(x_p^2 + 4 x_p+ 4) + (y_p^2 – 2y_p .(3/2) + 9/4) – R^2\} – \{ (x_p^2 + 2x_p+ 1) +$

$(y_p^2 – 2y_p (1/2) + (1/4) – R^2 \}$

$=x_p^2 + 4x_p+ 4+ y_p^2- 3y_p+ 9/4 - R^2 – x_p^2 - 2x_p - 1 – y_p^2 + y_p - 1/4 + R^2$

$= 2x_p + 3 – 2y_p + 8/4$

$\Delta d_{se} = 2x_p – 2y_p + 5$

- If $p_k < 0$
  - E is chosen, and the next midpoint will be one increment over in x.That is x= x+1

$\Delta d_E = p_{k+1} – p_k$

$= F(x_p + 1 + 1 ,y_p – 1/2) – F(x_p + 1, y_p – 1/2)$

$= F (x_p +2 , y_p - 1/2) – F(x_p + 1, y_p – 1/2)$

$= (x_p +2)^2 + (y_p - ½)^2 – R^2 – \{ (x_p + 1)^2 + (y_p – 1/2)^2 – R^2)\}$

$= (x_p +2)^2 + (y_p - ½)^2 – R^2 – (x_p + 1)^2 - (y_p – 1/2)^2 + R^2)$

$= (x_p +2)^2 – (x_p + 1)^2$

$= x_p^2 +4xp + 4 – (x_p^2 + 2xp + 1)$

$= x_p^2 +4xp + 4 – x_p^2 - 2xp - 1$

$\Delta d_E = 2x_p + 3$

- The initial decision parameter(P_0) is obtained by evaluating the circle function at the start position (x_0 , y_0) = (0, r)
  $P_0 = p_k$
  $= F(x_p + 1, y_p – 1/2)$

$= F(x_0 + 1, y_0 – 1/2)$

$=F (0 +1 ,R – 1/2)$

$= 1^2 + (R – ½)^2 – R^2$

$=1 + R^2 – R +1/4 – R^2$

$P_0 =5/4 – R$

- If the radius r is specified as an integer,we can simply round it to:

$P_0=1 – R$

Midpoint Circle Algorithm

- Midpoint circle algorithm generates all points on a circle centered at the origin by incrementing all the way around circle.
- The strategy is to select which of 2 pixels is closer to the circle by evaluating a function at the midpoint between the 2 pixels

Eight way symmetry

The shape of the circle is similar in each quadrant. Therefore, if we determine the curve positions in the first quadrant, we can generate the circle positions in the second quadrant of xy plane. The circle sections in the third and fourth quadrant can be obtained from sections in the first and second quadrant by considering the symmetry along X axis

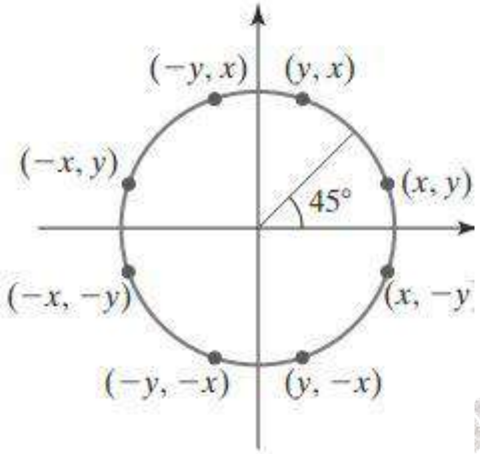
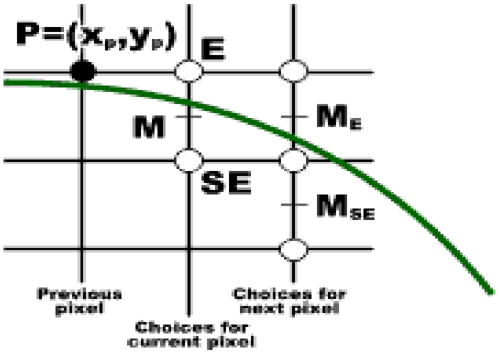


FIGURE 13
Symmetry of a circle. Calculation of a circle point (x, y) in one octant yields the circle points shown for the other seven octants.

  - Conside the circle centered at the origin,if the point (x, y) is on the circle,then we can compute 7 other points on the circle as shown in the above figure.
  - So we need to compute only one 45 degree segment to determine the circle completely.
- In **Midpoint Circle Algorithm** by evaluating a function at the midpoint between the 2 pixel,we can decide which pixel is closer to the circle.

- If pixel P at (x_p,y_p) has been chosen as the starting pixel,then the next pixel can be E or SE as shown in figure below.
- Now the choice is between E or SE



where $E= (x_{p+1}, y_p)$

$SE= (x_{p+1} , y_{p-1})$

$M=(x_{p+1} , y_p - 1/2)$

- According to implicit formula:

  $F(x,y) = x^2 + y^2 - R^2$, we use implicit formula for this algorithm.

- The relative position of any point (x ,y) can be determined by checking the sign of the circle function

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

- These tests are performed for the midpositions between pixels near the circle path at each sampling step.Thus the circle function is the decision parameter in the midpoint algorithm.
- If the midpoint **between** the pixel E and SE is outside the circle,then SE is closer to the circle
- If the midpoint is **inside** the circle, then pixel E is closer to the circle.
- As for lines,we choose on the basis of the decision variable d,which is the value of the function at the midpoint,

$$p_k = F(x_p+1 , y_p - 1/2)$$

$$= (x_p + 1)^2 + (y_p - 1/2)^2 - R^2$$

➢ If $p_k >= 0$

  - SE is chosen, where x is incremented by 1 and y is decremented by 1.

$$P_{k+1}=F(x_p + 2, y_p - 3/2)$$

$$\Delta d_{se} = p_{k+1} - p_k$$

$= F(x_p+2, y_p-3/2) – F (x_p+1 ,y_p – ½)$

$=(x_p + 2)^2 + (y_p – 3/2)^2 – R^2 – \{(x_p +1)^2 +$
$(y_p – ½)^2 - R^2$

Expand this equation using the formula $(a + b)^2$ and $(a - b)^2$

$=\{(x_p^2 + 4 x_p+ 4) + (y_p^2 – 2y_p .(3/2) + 9/4) – R^2\} – \{ (x_p^2 + 2x_p+ 1) +$

$(y_p^2 – 2y_p (1/2) + (1/4) – R^2 \}$

$=x_p^2 + 4x_p+ 4+ y_p^2- 3y_p+ 9/4 - R^2 – x_p^2 - 2x_p - 1 – y_p^2 + y_p - 1/4 + R^2$

$= 2x_p + 3 – 2y_p + 8/4$

$\Delta d_{se} = 2x_p – 2y_p + 5$

- If $p_k < 0$
    - E is chosen, and the next midpoint will be one increment over in x.That is x= x+1

$\Delta d_E = p_{k+1} – p_k$

$= F(x_p + 1 + 1 ,y_p – 1/2) – F(x_p + 1, y_p – 1/2)$

$= F (x_p +2 , y_p - 1/2) – F(x_p + 1, y_p – 1/2)$

$= (x_p +2)^2 + (y_p - ½)^2 – R^2 – \{ (x_p + 1)^2 + (y_p – 1/2)^2 – R^2)\}$

$= (x_p +2)^2 + (y_p - ½)^2 – R^2 – (x_p + 1)^2 - (y_p – 1/2)^2 + R^2)$

$= (x_p +2)^2 – (x_p + 1)^2$

$= x_p^2 +4xp + 4 – (x_p^2 + 2xp + 1)$

$= x_p^2 +4xp + 4 – x_p^2 - 2xp - 1$

$\Delta d_E = 2x_p + 3$

- The initial decision parameter(P_0) is obtained by evaluating the circle function at the start position (x_0 , y_0) = (0, r)
  $P_0 = p_k$
  $= F(x_p + 1, y_p – 1/2)$

$= F(x_0 + 1, y_0 – 1/2)$

$=F (0 +1 ,R – 1/2)$

$= 1^2 + (R – ½)^2 – R^2$

$=1 + R^2 – R +1/4 – R^2$

$P_0 =5/4 – R$

- If the radius r is specified as an integer,we can simply round it to:

$P_0=1 – R$

<u>Midpoint Circle Algorithm</u>

1. Input radius r and circle center (x_c , y_c), then set the coordinates for the first point on the circumference of a circle centered on the origin as

   $(x_0, y_0) = (0, r)$
2. Calculate the initial value of the decision parameter as

   $p_0 = 5 /4 − r$
3. At each x_k position, starting at k = 0, perform the following test: If $p_k < 0$, the next point along the circle centered on (0, 0) is (x_{k+1}, y_k) and

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

   Otherwise, the next point along the circle is ($x_{k + 1}$, $y_{k − 1}$) and

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

   where $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_{k − 2}$.

4. Determine symmetry points in the other seven octants.

5. Move each calculated pixel position (x, y) onto the circular path centered at (xc , yc) and plot the coordinate values as follows

Problem

Given a circle radius r = 10, we demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from x = 0 to x = y. The initial value of the decision parameter is

p0 = 1 − r = −9

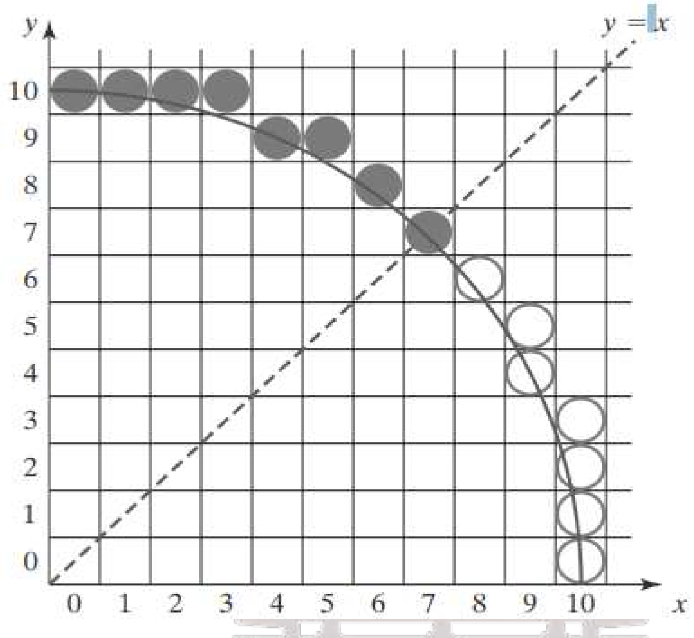
For the circle centered on the coordinate origin, the initial point is (x0, y0) = (0, 10), and initial increment terms for calculating the decision parameters are

2x0 = 0, 2y0 = 20

Successive midpoint decision parameter values and the corresponding coordinate positions along the circle path are listed in the following table:

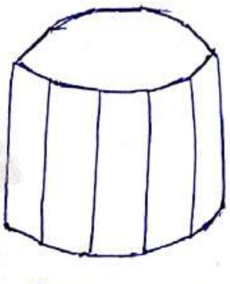
k	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	−9	(1, 10)	2	20
1	−6	(2, 10)	4	20
2	−1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	−3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7, 7)	14	14

A plot of generated pixel positions in the first quadrant is shown


y
y = x
10
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9
10
x

Fill-Area Primitives

- Describing components of a picture - is an area that is filled with some solid color or pattern. A picture component of this type is referred to as fill area or filled area.
- Approximating a curved surface with polygon facets is sometimes tessellation or fitting the surface with polygon mesh.



Wire frame representation for a cylinder

Polygon Fill Arears

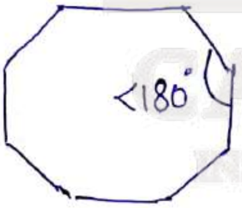
Polygon → is a plane figure specified by a set of 3 or more co-ordinate positions called vertices, that are converted in sequence by straight-line segments called edges or sides of the polygon.

Polygon must have all its vertices within a single plane and there can be no edge crossing (standard polygon or simple polygon).

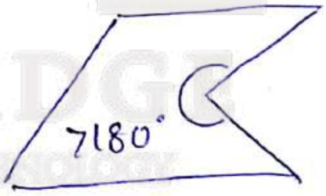
Problems → for a computer-graphics appl^n, it is possible that all the set of polygon vertices do not lie exactly in one plane. due to round off error in the calculation of numerical values, to errors in selecting co-ordinate positions of vertices or approximating a curved surfaced with a set of polygonal patches

Polygon classifications

- An interior angle → is an angle inside the polygon boundary that is formed by two adjacent edges.
- If the interior angle of a polygon are less than or = 180°, the polygon is convex
- A convex polygon is that its interior lies completly on one side of the infinite extension line of any one of its edges.
- A polygon that is not convex is called concave polygon
- The term degenerate polygon (rejected by graphics packages) is often used to describe a set of vertices that are collinear (generating a line segment) or that have repeated co-ordinate positions - can generate a polygon shape with extraneous lines, overlapping edges or edges that have a length equal to 0.



Convex polygon



Concave polygon

Problems with concave polygons

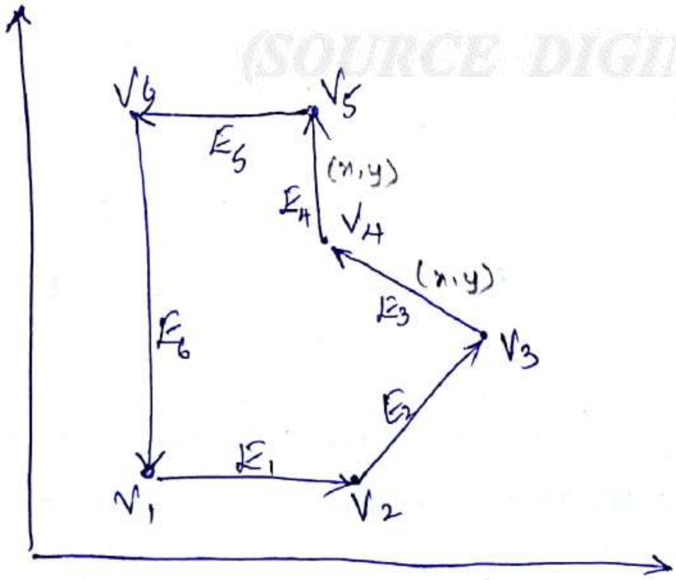
1) Implementing fill algorithms & other graphics routines are more complicated → so it is generally more efficient to split a concave polygon into a set of convex polygons before processing.

2) concave polygon splitting is often not included in graphics library

Some graphics packages, including OpenGL, requires all fill polygons to be convex.

Identifying Concave Polygons.

- → A concave polygon has at at least one interior angle greater than 180°.
- → extension of some edges of a concave polygon will intersect other edges, some pair of interior points will produce a line segment that intersects a polygon boundary.
- → set up vectors for all edges
- → Perform cross product to adjacent vectors to test for concavity.
- → Perform dot product if we want to determine the angle between two edges.
- → All vector products will be same value (+ve or -ve) for convex polygon.
- → If there are some cross products yield a positive & some yield negative value, we have a concave polygon.



$(E_1 \times E_2)_z > 0$

$(E_2 \times E_3)_z > 0$

$(E_3 \times E_4)_z < 0$

$(E_4 \times E_5)_z \cancel{>} 0$

$(E_5 \times E_6)_z > 0$

$(E_6 \times E_1)_z > 0$

$E_{3x} E_{4y} - E_{3y} E_{4x}$

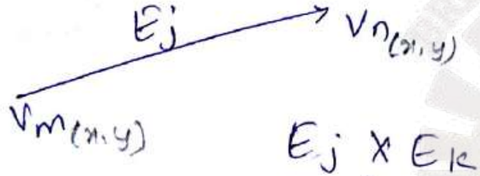
Splitting Concave Polygons.

→ Splitting can be accomplished using edge vectors & edge cross products.

(OR)

we can use vertex positions relative to an edge extension line to determine which vertices are on one side of this line & which are on the other.

→ Vector method for splitting a concave polygon.



Dot product

$E_j = (V_{nx} - V_{mx}, V_{ny} - V_{my})$

$E_j \times E_k = E_{jx} E_{ky} - E_{jy} E_{kx}$

for 2 successive edges vectors is a vector perpendicular to xy plane with z component equal to

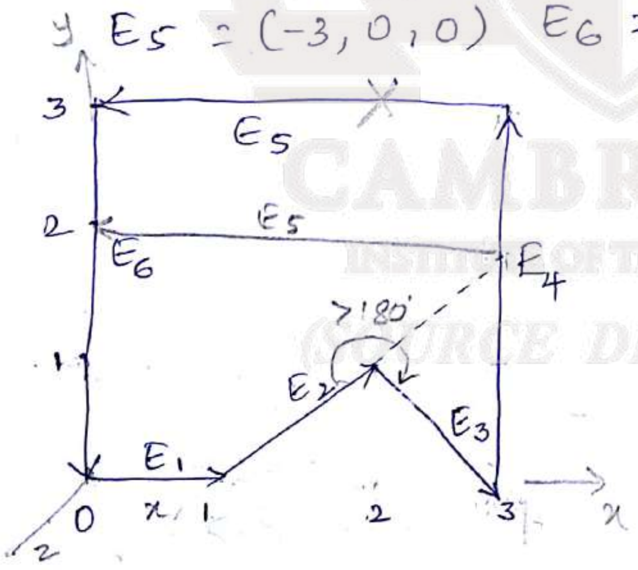
Ex: Vector Method. - concave polygon with six edges

$\overset{E_j}{E_1} = (\underset{x}{1}, \underset{y}{0}, \underset{z}{0})$ $\quad \overset{E_k}{E_2} = (\underset{x}{1}, \underset{y}{1}, \underset{z}{0})$

$E_3 = (\underset{x}{1}, \underset{y}{-1}, \underset{z}{0})$ $\quad E_4 = (0, 2, 0)$

$E_5 = (-3, 0, 0)$ $\quad E_6 = (0, -2, 0)$

since z component is '0', all edges are in xy plane



$E_{jx} E_{ky} - E_{kx} E_{jy}$

1-0 =

$E_1 \times E_2 = (0, 0, 1)$ ✓

$E_2 \times E_3 = (0, 0, -2)$ ✓ $\quad -1-1=-2$

$E_3 \times E_4 = (0, 0, 2)$ $\quad 2-0=2$

$E_4 \times E_5 = (0, 0, 6)$

$E_5 \times E_6 = (0, 0, 6)$

$E_6 \times E_1 = (0, 0, 2)$

→ Since $E_2 \times E_3$ is negative z component, we split the polygon ~~line~~ along the line of vector E_2

→ line eq^n of this edge has slope 1 & y intercept -1

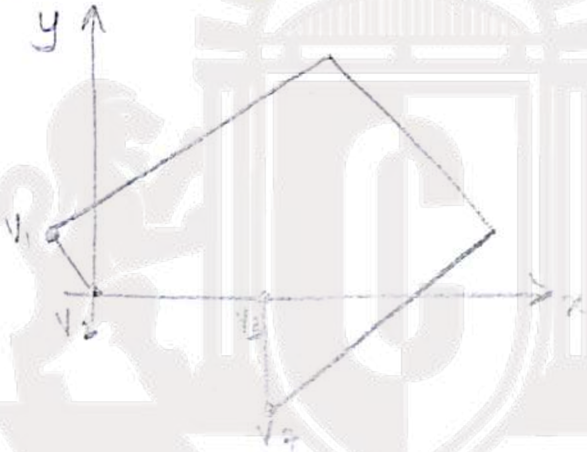
→ hence polygon edges split the polygon into two pieces.

→ No other edge cross product is negative, so two new polygons are convex.

Splitting a convex polygon into a set of triangles

→ Once a convex polygon is obtained with a vertex list, we could transform into a set of triangles.

→ any sequence of three consecutive vertices are formed to be a new polygon (a triangle)



Rotational Method - for concave polygons

→ shift the position of polygon, vertex V_k at the co-ordinate origin.

→ rotate the polygon about the origin in ~~anti~~clock wise so that the next vertex V_{k+1} is on the x axis.

→ if polygon ~~is~~ has vertex V_{k+2} below x-axis the polygon is concave.

Inside-Outside Tests.

→ With complex objects, we may have to specify a complex fill region with intersecting edges.

→ for such shapes, we must decide how to determine whether a given point is inside or outside the polygon.

→ conceptually, the process of filling the inside of a polygon with a color or pattern is equivalent to deciding which points in the plane of the polygon are interior (inside) points or exterior (outside).

There are two types of test.
i) Crossing (or) Odd - even test
2) Winding number test

1. Odd-Even test

This test is most widely used for making inside-outside decisions.

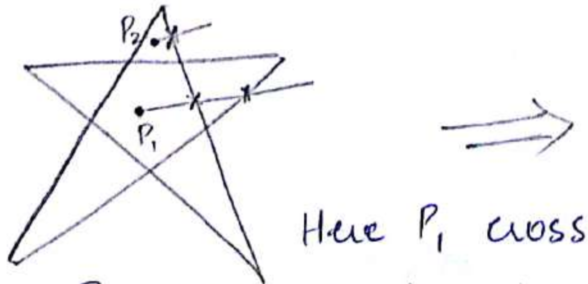
→ Drawing a line from any position 'P' (inside) to a distant point outside the co-ordinate extents of the closed polyline.

→ count the number of line-segment crossing's along this line

→ If the number of segments crossed by this line is odd - P is considered to be interior point

→ otherwise P is exterior.

Ex:-



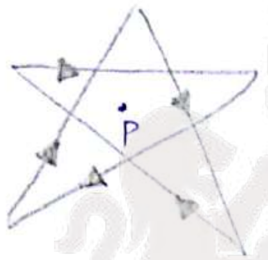
Here P_1 crosses 2 edges, hence Outside or exterior. P_2 crosse 1 edge, hence interior/inside

2) winding number Test [Non-zero winding number rule]

This test fills the complete star rather than in previous test

→ To implement this test, we consider traversing the edges of the polygon from any starting vertex and going around the edge in a particular direction(~~which~~ any direction) until we reach the starting point.

→ we illustrate the path by labeling the edges, as shown in the below fig



▷ → labeling the edges

→ consider an arbitrary point. Initial the winding number is set to 'zero'.

→ winding number, which counts the number of ~~objects~~ times the boundary of an object "winds" around a particular point in counterclockwise direction.

→ count clockwise as positive(+1) or add 1 to windowing number when it intersects a segment that crosses the line in clockwise direction

→ count anti counterclockwise as -1. negative

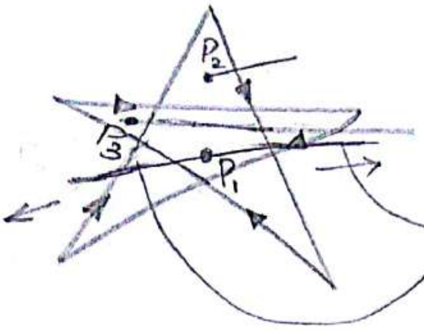
→ If windowing number is non-zero, P is Interior

If windowing number is zero, P is exterior

→ All ~~edges~~ points must cross edges not vertices

(cont)

Ex 1.



⇒ P_1 crosses 2 edges,
1st is from right to left.

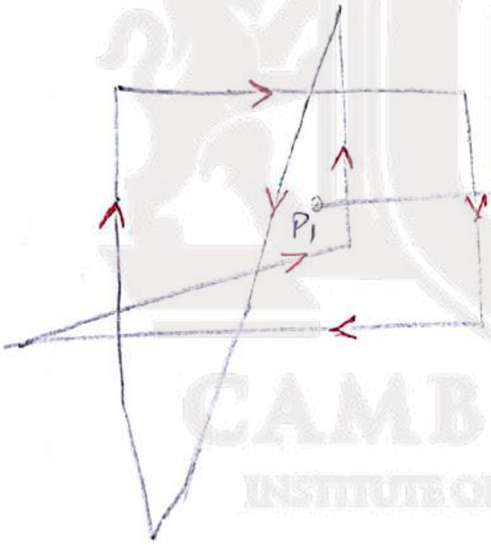
$P_1 = +1+1 = 2$ (Inside)
from vertex P_1 to left

$P_1 = -1-1 = -2$ (Inside)
from vertex P_1 to right

⇒ P_2 = crosses 1 edge from right to left
$P_2 = +1$ (Inside)

⇒ P_3 = crosses 3 edge.
$= -1+1+1$
$P_3 = 1$ (Inside)

Ex 2.



$P_1 = -1+1$
$P_1 = 0$ (outside)

Polygon tables

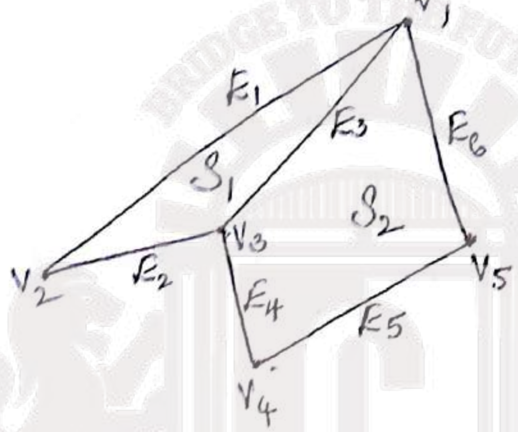
→ The description of objects includes co-ordinate informatn specifying the geometry for the polygon facets and other surface parameters such as color, transparency, light-reflection properties

→ As information for each polygon is input, the data are placed into polygon tables for further processing, display & manipulation of objects

Polygon data tables are organised into 2 groups

① Geometric tables contain vertex co-ordinates & parameters to identify the spatial orientation of polygon surfaces.

* a vertex table & edge table & surface-facet table



Vertex table

V_1 : x_1 y_1 z_1
V_2 : x_2 y_2 z_2
V_3 : x_3 y_3 z_3
V_4 : x_4 y_4 z_4
V_5 : x_5 y_5 z_5

Edge table

E_1 : V_1 V_2
E_2 : V_2 V_3
E_3 : V_3 V_1
E_4 : V_3 V_4
E_5 : V_4 V_5
E_6 : V_5 V_1

Surface-facet table

S_1 : E_1 E_2 E_3
S_2 : E_3 E_4 E_5 E_6

Vertex table + Surface-facet table → is less convinient & some edges could be drawn twice in a wire-frame display.

Surface-facet table → duplicates co-ordinate information

E_1 : V_1 V_2 S_1
E_2 : V_2 V_3 S_1
E_3 : V_3 V_1 S_1 S_2
E_4 : V_3 V_4 S_2
E_5 : V_4 V_5 S_2
E_6 : V_5 V_1 S_2

Edge table expanded to include pointers into surface-facet table

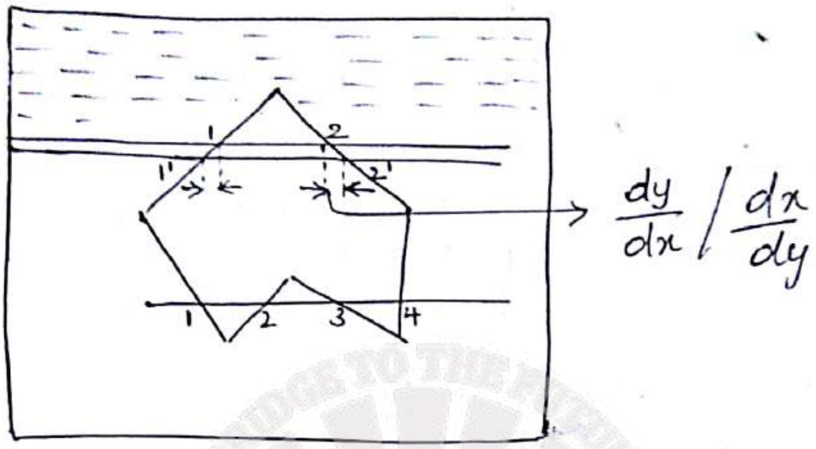
There is importance to check the consistency & completeness of data.

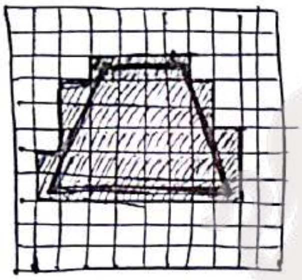
Some tests that could be performed by graphics packages are

1) that every vertex is listed as an endpoint for atleast 2 edges
2) that every edge is part of at least one polygon
3) that every polygon is closed
4) that each polygon has at least one shared edge
5) that if the edge table contains pointers to polygons, every edge referenced by a polygon pointer has a reciprocal pointer back to polygon

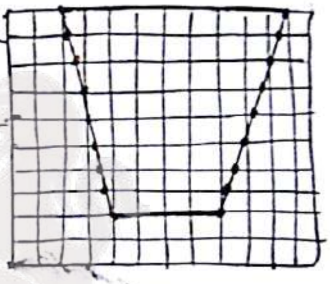
Scanline Algorithm / scan Conversion

General Scan-Line Polygon-fill Algorithm





Vertices forming polygon, aliasing is seen Pixels are at the center of the grid



Pixels are not at the center of the grid, but at the intersection of two orthogonal Scan lines (on grid intersection points)

→ Not drawing lines, but filling regions

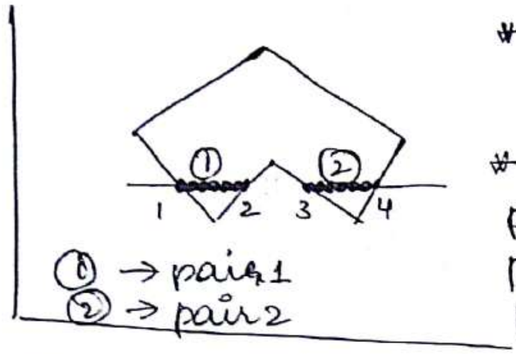
Steps

1) when a polygon is defined, find a minimum enclosed rectangle (binds a polygon) - find the x_{min} & x_{max}, y_{min} & y_{max} where all the vertices falls with minimum area that covers a polygon

2) find the no of Scanlines $= y_{max} - y_{min} + 1$

3) for each of scanlines do
   - obtain the intersection points of Scanline with polygon edges
   - sort the intersection points from left to right, identify interior regions
   - as the odd - even rule
   - form pair of intersections from the list

- fill colors are applied to each section pair of a scanline that lies within the interior of fill region

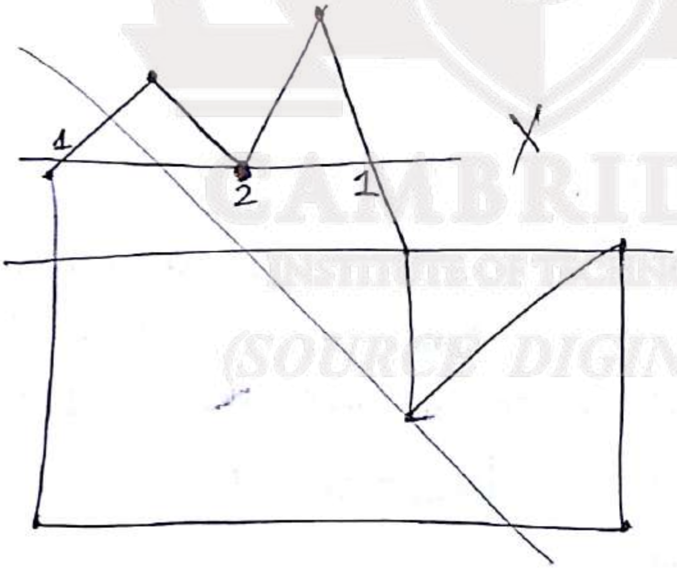


- No of intersections are Even no's forming pairs
- Look left or right (pixel inside the pair), count the no of intersection points of the scanline with edge of polygon, odd → interior else exterior

- Intersection points are updated for each scanline
- Stop when scanline has reached Ymax

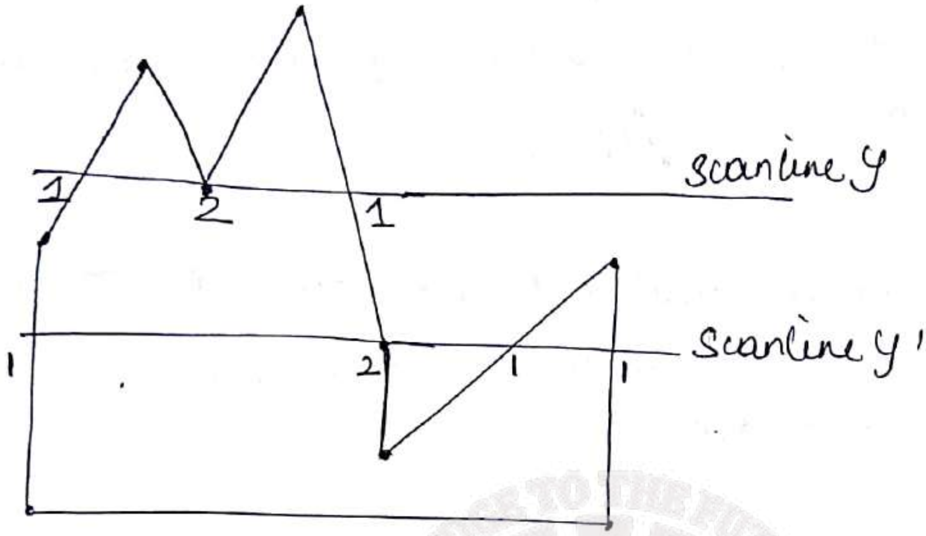
Special issues

Whenever a scan line passes through a vertex, it intersects two polygon edges at that point.
This can result in an odd number of boundary intersections for a scanline.



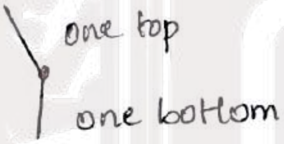
- Introduction - when we move from one scanline to another 1) $\frac{dy}{dx}$, intersection points change by $\frac{\Delta y}{dx}$.
  2) New edge may start / End.

- If we know the intersection points, then all the next points are calculated by the slope of the line



for y, the edges at the vertex are on the same side of the scanline V bottom ∧ top

Whereas for y', the edges are on either/both sides of the vertex



(opposite sides of scanline)

In such case additional preprocessing is required

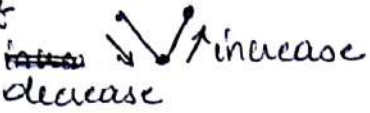
Vertex counting in a scanline

- Traverse along the polygon boundary clockwise or counter clockwise
- observe the relative change in y-value of the edges on either side of the vertex (i.e as we move from one edge to another)

Check for condition

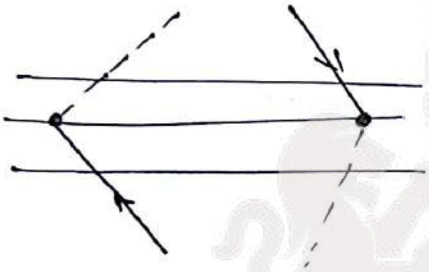
If end-point y values of two consecutive edges monotonically increase or decrease, } ↓ decreasing clockwise /increasing anticlock count the middle vertex as a single intersection point for the scanline passing thought it.

Else the shared vertex represents a local maximum (or minimum) on the polygon boundary. Increment the intersection count



if the vertex is a local extrema, consider (or add) 2 intersections for the scan line corresponding to such a shared vertex.

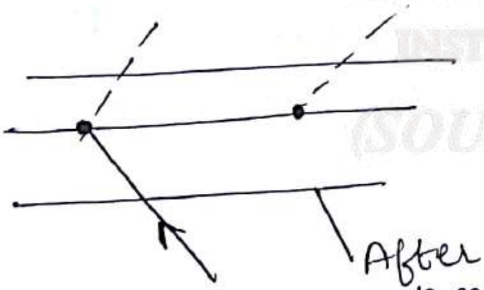
Must avoid such cases listed below.



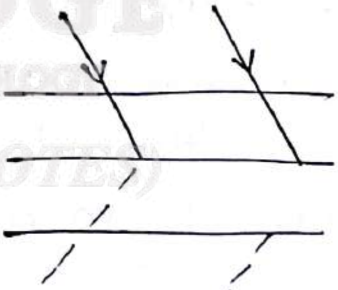
should not add intersection point.

To implement the above.

while processing non-horizontal edges along a polygon boundary in any order, check to determine the condition of monotonically changing (increasing or decreasing) endpoints y values



Before processing After processing

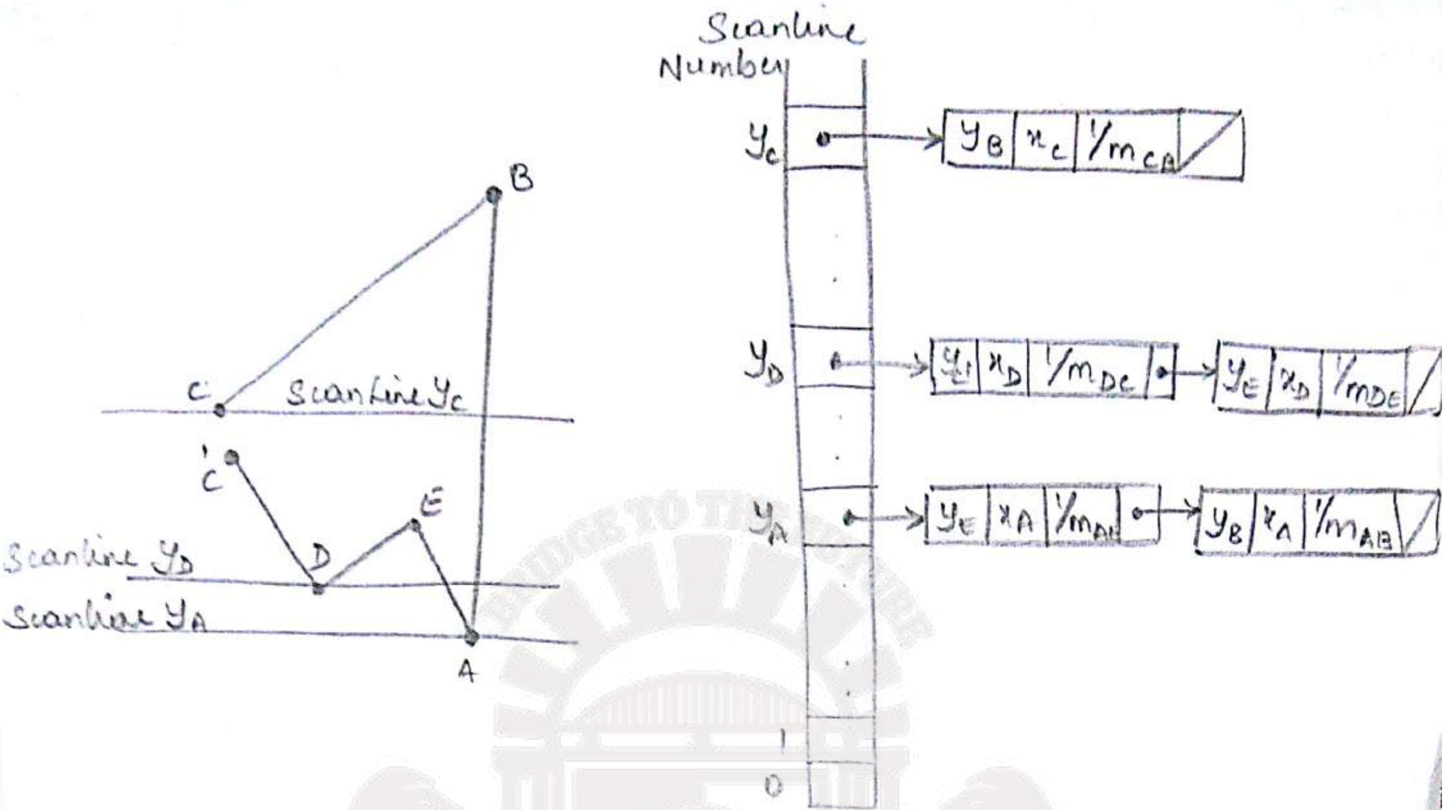


Before processing After processing

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}.$$

y-coordinate b/w 2 scan lines $y_{k+1} - y_k = 1$. x-intersection value x_{k+1} can be determined by

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



- * To efficiently perform a polygon fill, store the polygon boundaries in a sorted edge table containing all information necessary to process the scanlines efficiently.
- * use Buckets sort to store edges, sorted on smallest y value of each edge.
- * Only Non-horizontal edges are entered into the sorted edge table.
- * As edges are processed, we can also shorten certain edges to resolve the vertex-intersection question.
- * each table entry contains (for a particular scanline) the maximum y value for that edge, the x-intercept value (at lower vertex) for the edge, & inverse slope of the edge.
- * we process the scan lines later from the bottom of the polygon to its top producing an {active edge list} for each scan line crossing the polygon boundaries.

will contain all edges crossed by scanline, used to obtain edge intersections.

OpenGL Polygon fill-Area functions

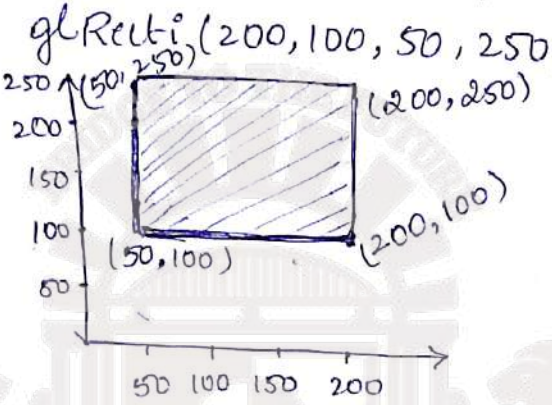
→ glRect* $(x_1, y_1, x_2, y_2$

One corner of rectangle is at co-ordinate position (x_1, y_1) and the opposite corner of the rectangle at position (x_2, y_2).

→ Suffix codes for glRect specifies the co-ordinate data type and whether co-ordinates are to be expressed as array elements.

- i - integer
- s - short
- f - float
- d - double
- v - vector

glRecti (200, 100, 50, 250);



if we put co-ordinate values for this rectangle into Arrays we generate a same square with following code

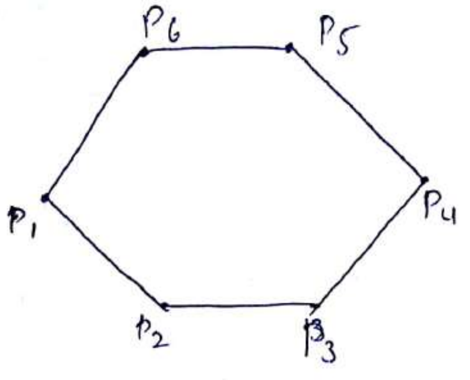
```
int vertex1 [ ] = (200,100)   (x1, y1)
int vertex2 [ ] = (50, 250)   (x2 y2)
glRectiv( vertex1, vertex2);
```

when glRect is used, the polygon edges are formed between the vertices in order $(x_1, y_1), (x_2, y_1), (x_2, y_2) (x_1, y_2)$

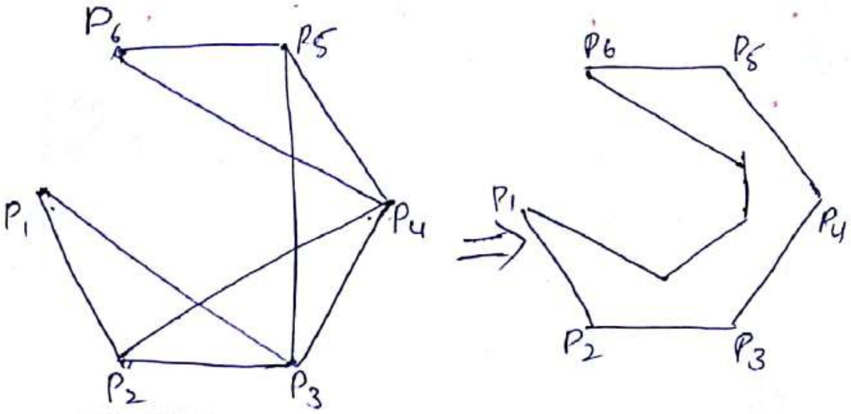
(200, 100), (50, 100), (50, 250), (200, 250)

```
* glBegin (GL-POLYGON):
      glVertex2iv (p1);
      glVertex2iv (P2);
      glVertex2iv (P3);
      glVertex2iv (P4);
      glVertex2iv (P5);
  glEnd ();
```

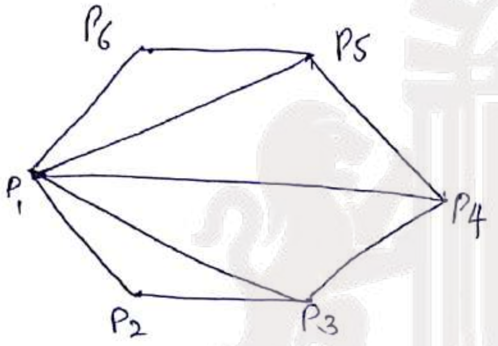
* polygon vertex list must contain at least three Vertices, otherwise nothing will be displayed.



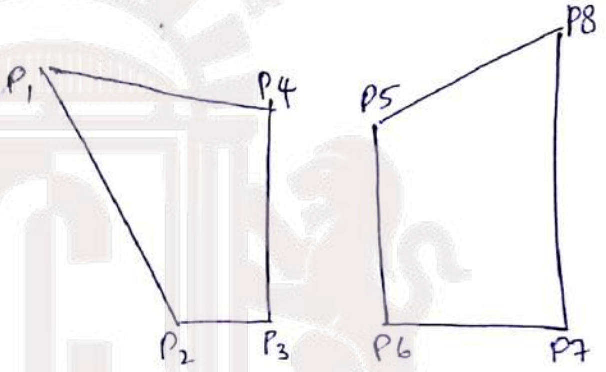
GL_POLYGON



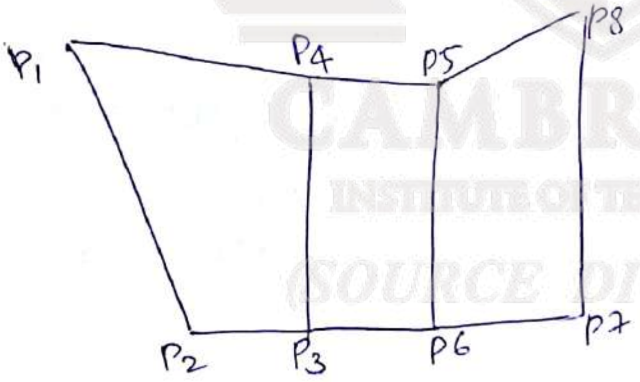
GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP

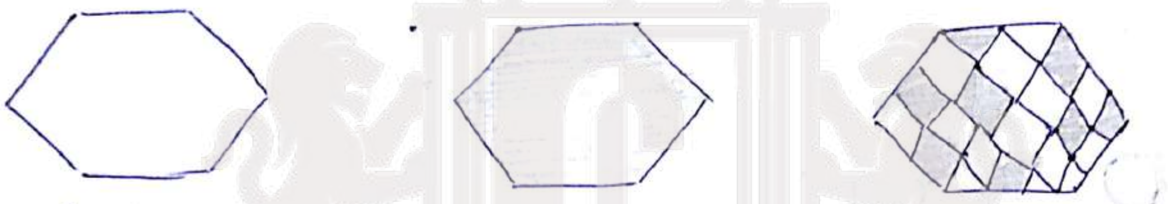
Fill area Attributes

There are two basic procedures for filling an area on raster systems.

① determine the Overlap intervals for scan lines that cross the area, then pixel positions along these overlap intervals are set to fill color.

② start from interior position & paint outward, pixel by pixel, from this point until we encounter specified boundary conditions

Fill Styles



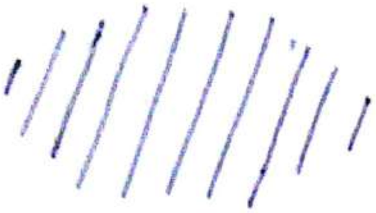
a) Hollow b) Solid c) Pattern

We can Also fill selected regions of scene using brush styles, color-blending combinations or textures.

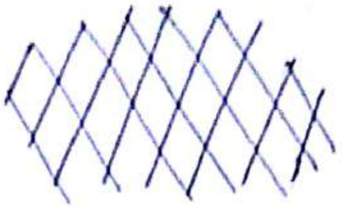
we can also list different colors for different positions in the array / OR fill pattern could specify the bit array that indicates which relative positions are to be displayed in a single selected color.

* An array specifying a fill pattern is a mask that is to be applied to display area.

* the process of filling an area with a rectangular pattern is called ~~tiling~~ "tiling" & the rectangular fill pattern is sometimes refered as "tiling pattern"



Diagonal Hatch fill

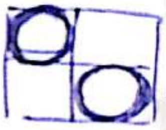


Diagonal crosshatch fill

Spacing & slope for the hatchlines could be set as parameters in <u>hatch table</u> or specified as pattern array that produces sets of diagonal lines.

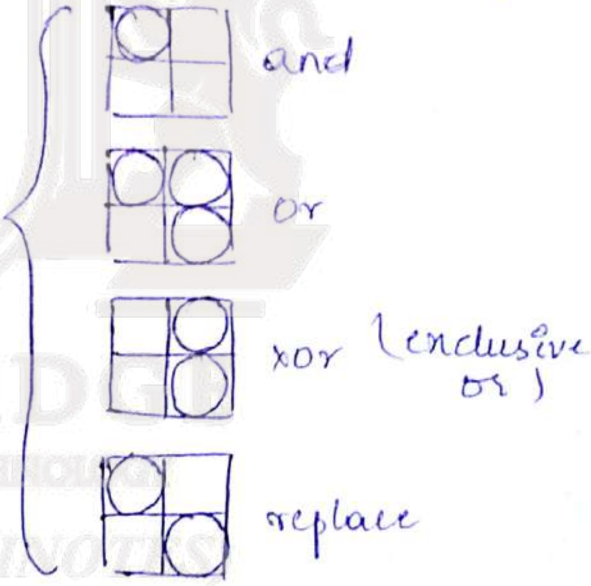
Color - Blended fill regions

- combine a fill pattern with background colors (Pattern) using a transparency factor that determines how much of background should be mixed with object color.



Pattern

Background



Pixel values

Application

Fill methods using blended colors have been referred to as soft-fill or tint fill algorithms.

① Usage is to soften the fill colors at object borders that have been blurred to antialias the edge

② is to allow repainting of a color area that was originally filled with a semitransparent brush, where the current color is then a mixture of brush color & background color "behind" the area.

Linear soft-fill algorithm repaints an area merging a foreground color F with a single background color B, where F=B.

Assume we know the values of F and B, we can check the contents of frame buffer to determine how these colors were combined.

current RGB color P of each pixel

$$P = tF + (1-t)B.$$

t → transparency factor between 0 & 1 for each pixel

t < 0.5, background color contributes more to interior color of region than the fill color does.

$P = (P_R, P_G, P_B)$ $F = (F_R, F_G, F_B)$ $B = (B_R, B_G, B_B)$

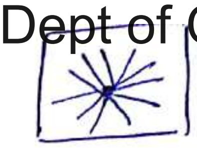
$$t = \frac{P_K - B_K}{F_K - B_K}$$

- color blending procedures can be applied to an area, foreground color merged with multiple background color areas,

when two background color B_1 & B_2 are mixed with foreground color F, resulting pixel color P

$$P = t_0 F + t_1 B_1 + (1 - t_0 - t_1) B_2.$$

color co-efficient t_0, t_1 & $(1-t_0-t_1)$ must be equal to 1.



∞ extent

Plane Equations.

Each polygon in a scene is contained within a plane of infinite extents. The general equation of plane is

$$Ax + By + Cz + D = 0 \rightarrow ①$$

where x, y, z is any point on the plane.

A, B, C, D are plane parameters – describing spatial properties of plane.

To obtain A, B, C & D values, three plane equations are solved by using co-ordinate values with 3 non collinear points. $(x_1 y_1 z_1)$ $(x_2 y_2 z_2)$ $(x_3 y_3 z_3)$

$$\div \text{ by } D \rightarrow ① \qquad \begin{vmatrix} + & - & + \\ - & + & - \\ + & - & + \end{vmatrix}$$

$$\left(\frac{A}{D}\right)x_k + (B/D)y_k + (C/D)z_k = -1 \qquad k = 1, 2, 3$$

Solution to eqn is obtained using cramer's rule.

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

$$A = -y_1(z_3 - z_2) + y_2(z_3 - z_1) - y_3(z_2 - z_1)$$

$$= y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2)$$

$$B = z_1(x_2 - x_3) - z_2(x_1 - x_3) + z_3(x_1 - x_2)$$

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

$$= z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2)$$

Expanding determinants we get the below

$$A = y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2)$$

$$B = z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2)$$

$$C = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$$

$$D = -x_1(y_2 z_3 - y_3 z_2) - x_2(y_3 z_1 - y_1 z_3) - x_3(y_1 z_2 - y_2 z_1)$$

Front and Back Polygon faces

* The side of a polygon that faces into the object interior is called backface.

* the visible, or outward side is front face.

* Identifying the position of points in space relative to front & back faces of a polygon – Basic task.

A polygon with infinite plane.

* Any point that is not on the plane & that is visible to the front face → in front of (outside the object)

* Any point that is visible to the back face of polygon is behind (inside) the plane.

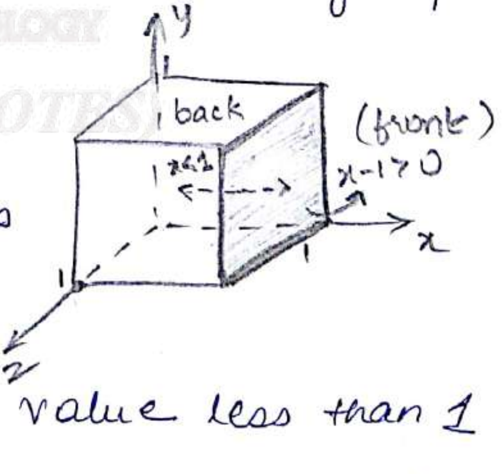
* Inside / outside is relative to the plane containing the polygon.

plane equation → $Ax+By+Cz+D \neq 0$. (Not on plane)

if $Ax+By+Cz+D < 0$ the point (x,y,z) is behind the plane.

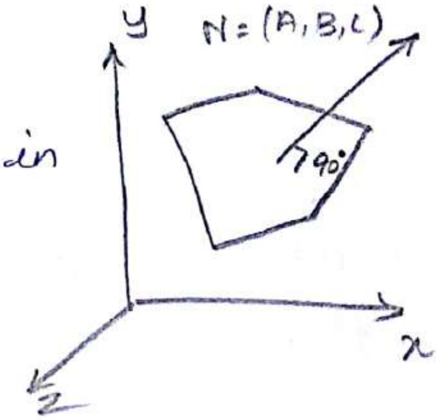
If $Ax+By+Cz+D > 0$ the point (x,y,z) is in front of the plane

Any point outside (front of) the plane of shaded polygon satisfies the inequality $x-1>0$
while any point inside the plane (in back of) has x co-ordinate value less than 1



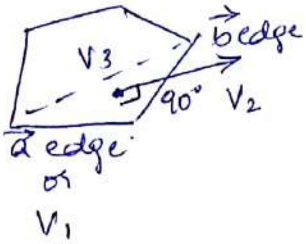
* Orientation of a polygon surface in space can be described with normal vector (perpendicular to plane) & has cartesian components A, B, C (plane co-efficients)

- Normal vector points in direction from inside the plane to outside i.e from back face of polygon to front face.



- The normal vector N = (1, 0, 0) – is in the direction of positive x-axis

Suppose.



$$N = (V_2 - V_1) \times (V_3 - V_1)$$

- This generates values for plane parameters A, B & C
- Elements of normal vector can be obtained using vector cross product calculation.
- Plane equation in vector form is $N.P = -D$

  N → normal vector, P is any point in the plane.

OpenGL Fill-Area Attribute functions

Displays of filled convex polygon in 4 steps.

1. Define a fill pattern.

GLubyte fillPattern [ ] = { 0xFF, 0x00, 0xFF, 0x00...};

* to fill the patter in OpenGL, we use a 32bit x 32 bit mask.
* Value 1 in mask indicates the corresponding pixel is to be set to the current color.
* Value 0 → leaves the value of that frame buffer position unchanged.

2. Invoke the polygon fill routine

glPolygonStipple (fillPattern);

we need to enable the fill routines before we specify the vertices for the polygons that are to be filled with the current patter. hence ③.

3. we activate the polygon-fill feature of OpenGL

glEnable (GL_POLYGON_STIPPLE);

we turn off pattern filling with

glDisable (GL_POLYGON_STIPPLE);

4. Describe the polygons to be filled

OpenGL Texture & Interpolation Patterns

* Use texture patterns to fill polygons.
* Similar to ~~Similar~~ Simulate the surface appearances of wood, brick, brushed steel ...
* Interpolation fill of a polygon interior is used to produce realistic displays of shaded surface under various lighting conditions.

```
glShapeModel(GL_SMOOTH);
glBegin(GL_TRIANGES);
    glColor3f(0.0, 0.0, 1.0);
    glVertex2i(50, 50);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2i(150, 50);
    glColor3f(1.0, 0.0, 0.0);
    glVertex2i(75, 150);
glEnd();
```

GL_FLAT → fills the polygon with one color.

GL_SMOOTH → default shading.

OpenGL Wire-frame Methods.

- to show only polygon edges (produces wire frame or hollow display of polygon).

glPolygonMode (face, displayMode);

\# parameter 'face' which face of polygon we wante to show edges. GL_FRONT, GL_BACK, GL_FRONT_AND_BACK

\# Display Mode → GL_LINE.
GL_POINTS (polygon vertex points)

- Stitching → methods for displaying the edges of a filled polygon may produce gaps along the edges. due to scanline fill or edge line-drawing algo calculation.

- to eliminate the gap - shift the depth values calcu-

-lated by the fill routine so that they do not overlap with edge depth values for that polygon

glColor3f (0.0, 1.0, 0.0);
glEnable(GL_POLYGON_OFFSET_FILL); - set routine for scanline filling
glPolygonOffset (1.0, 1.0);
glDisable (GL_POLYGON_OFFSET_FILL);

glPolygonOffset (factor1, factor2);

calculate amount of depth offset.

$$\text{depthoffset} = \text{factor1} \cdot \text{maxSlope} + \text{factor2} \cdot \text{const}$$

→ GL_POLYGON_OFFSET_LINE
→ GL_POLYGON_OFFSET_POINT

- To eliminate selected edges from wire-frame display - glEdgeFlag (flag);

This indicates that a vertex does not proceed a boundary edge, GL_FALSE to parameter flag.

OpenGL Front face function.

Although the ordering of polygon vertices controls the identification of front & back faces.

we can label the selected faces in the scene independently as front or back with the function.

glFrontFace (vertexorder);

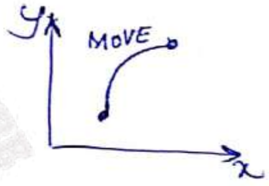
The vertexOrder in OpenGL when set to GL_CW (clock-wise ordering) for its vertices will be considered to the front face.

If the vertex order in OpenGL, GL-CCW (counter clockwise ordering) of polygon vertices as front-facing which is the default ordering.

Basic Two-Dimensional Geometric Transformations.

Representation of points:

$\overset{R}{2} \times \overset{C}{1}$ matrix $\begin{bmatrix} x \\ y \end{bmatrix}$



General Method of applying transformation.

geometric transform matrix / Affine transformation matrix

$$[B] = [T][A]$$

Transformed co-ordinates xprime, yprime

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \overrightarrow{\begin{bmatrix} a & c \\ b & d \end{bmatrix}} \begin{bmatrix} x \\ y \end{bmatrix} \downarrow \qquad \begin{bmatrix} x' \\ y' \end{bmatrix}^T = \begin{bmatrix} x \\ y \end{bmatrix}^T \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$x' = ax + cy$ | $x' = ax + cy$

$y' = bx + dy$ | $y' = bx + dy$

Premultiplication | Post Multiplication.

Special cases of 2D Transformations.

1) T = identity matrix

$a = d = 1, \; b = c = 0 \Rightarrow x' = x, \; y' = y.$

2) Scaling & Reflections:

$b = 0$, ~~&~~ $c = 0 \Rightarrow x' = a.x, \; y' = d.y;$

This is scaling by a in x, d in y.

if $a = d > 1$, we have enlargement (zooms)

if $0 < a = d < 1$, we have compression (reduction)

Scaling is uniform. if a & d value are same

" non-uniform; if a & d do not have same/identity value i.e if $a = d \rightarrow$ uniform scaling else non-uniform scaling.

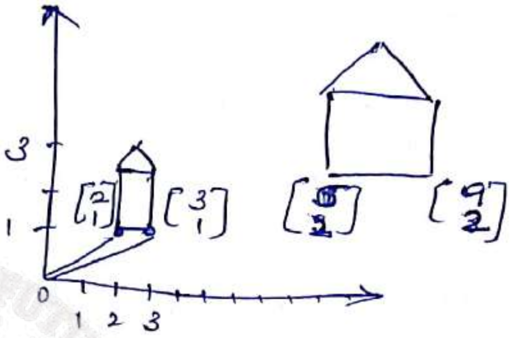
Scale Matrix : let $s_x = a$, $s_y = d$ $\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$

Example of scaling

Non-Uniform

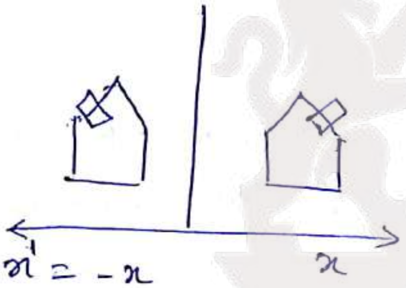
$s_x = 3$

$s_y = 2$



Only the diagonal terms are involved in scaling and reflections

Some more examples. (Reflection)



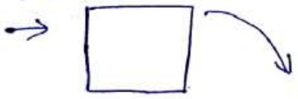
we will be considering the diagonal value. $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ b & c = 0.

In this case x will be negative y remains same +ve.

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

off diagonal elements are involved in shearing

Force is applied



shears along the axis

Very huge dictionary

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

$a = d = 1$;

let $c = 0$, $b = 2$

$x' = x$

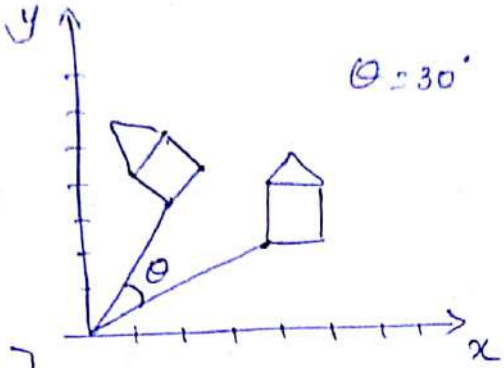
$y' = 2x + y$

$x' = ax + cy$

$y' = bx + dy$.

y' depends linearly on x : This effect is called shear.

Rotation

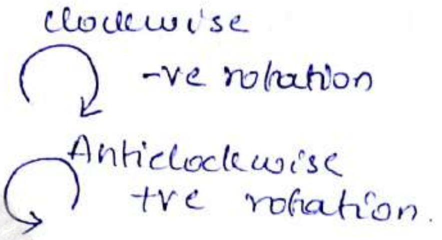


$x' = x\cos(\theta) - y\sin(\theta)$

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

matrix form is

$$T = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$



For rotation determinant $|T| = 1$

$\cos^2\theta + \sin^2\theta = 1$

& $|T|^T = |T|^{-1}$ (Transpose of matrix is inverse.)

* Rotation matrices are orthogonal

Two-Dimensional Translation

- A Perform translation on a single co-ordinate point by adding offsets to co-ordinates so as to generate a new co-ordinate position.
- In effect, we are moving the point along a straight line path to its new location.
- ||ly an entire object is moved (multiple co-ordinates) by rotating all the co-ordinate positions by same displacement along parallel paths. Then the complete object is displayed at new location.
- translation distance t_x and t_y to original co-ordinates (x, y) to obtain new co-ordinate positions (x', y')

$$x' = x + t_x, \quad y' = y + t_y$$

(t_x, t_y) is translation vector or shift vector.

matrix format.

$P = \begin{bmatrix} x \\ y \end{bmatrix}$ $P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ $T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$

$P' = P + T.$

Translation is a rigid-body transformation that moves objects without deformation (every point on object is translated by same amount).

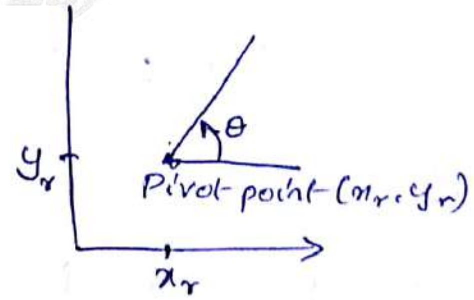
Two Dimentional Rotation.

- Rotation transformation of an object is done by specifying a rotation axis and a rotation angle
- All points of objects are then transformed to new positions by rotating points through specified angle about the rotation axis.
- Parameters of 2D rotation are the rotation angle θ, position (x_r, y_r) called rotation point or pivot point (intersection point/position of rotation axis ~~of~~ with xy plane).
- +ve value for angle θ → counterclockwise rotation
- -ve value - clockwise.

Formula in Trignometry

$\cos\theta = \frac{\text{adjacent side}}{\text{hypotenous}}$

$\sin\theta = \frac{\text{opposite side}}{\text{hypo}}$



$\cos(A+B) = \cos A \cos B - \sin A \sin B$

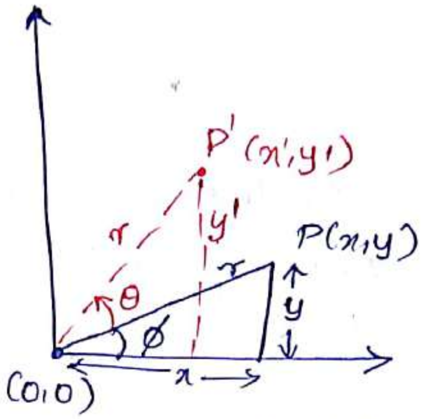
$\sin(A+B) = \sin A \cos B + \cos A \sin B$

$\cos(A-B) = \cos A \cos B + \sin A \sin B$

$\sin(A-B) = \sin A \cos B - \cos A \sin B$

$\cos\phi = \frac{x}{r} \Rightarrow x = r\cos\phi$

$\sin\phi = \frac{y}{r} \Rightarrow y = r\sin\phi$



Rotation in Anticlockwise wise

The new angle after rotation from $P + P' = (\phi+\theta)$

so $\cos(\phi+\theta) = \frac{x'}{r}$

$\boxed{x' = r.\cos(\phi+\theta)} = r[\cos\phi \cos\theta - \sin\phi . \sin\theta]$

$= r\cos\phi \cos\theta - r.\sin\phi.\sin\theta$.

$\boxed{x' = x\cos\theta - y\sin\theta} \rightarrow ①$

$\boxed{\because x = r\cos\phi \quad y = r\sin\phi}$

||ly $\sin(\phi+\theta) = \frac{y'}{r}$

$\boxed{y' = r.\sin(\phi+\theta)} = r.\sin\phi \cos\theta + \cos\phi.\sin\theta$

$= r.\sin\phi \cos\theta + r.\cos\phi.\sin\theta$.

$\boxed{y' = y\cos\theta + x\sin\theta} \rightarrow ②$

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

$\therefore P' = R.P$

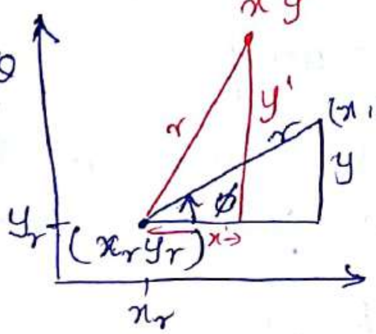
$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

If we solve this matrix we will get eqn ① & ②

Rotating a point from position (x, y) to position (x', y') through an angle θ about the rotation point (x_r, y_r)

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

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



Two-Dimensional Scaling

- To alter size of the object, we apply a scaling transformation
- scaling operation is performed by multiplying object positions (x, y) by scaling factors S_x and S_y to produce

Let us consider

$P = (x, y) \rightarrow$ before scaling &

$P'(x', y') \rightarrow$ after scaling

$x' = x . Sx$

$y' = y . Sy$

$$\therefore \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \begin{matrix} x' = x . S_x \\ y' = y . S_y \end{matrix}$$

$$P' = S . P .$$

- if S_x & S_y is in between 0 & 1, then point is closer to origin which means the size of an object will decrease
- If S_x & S_y are greater than 1, then the point is away from the origin, which means the size of an object ~~de~~ increases.
- If S_x & S_y are equal, assigned to same value,

uniform scaling is performed.

Unequal values for S_x & S_y result in differential scaling

we can control the location of a scaled object by choosing a position, called fixed position/point

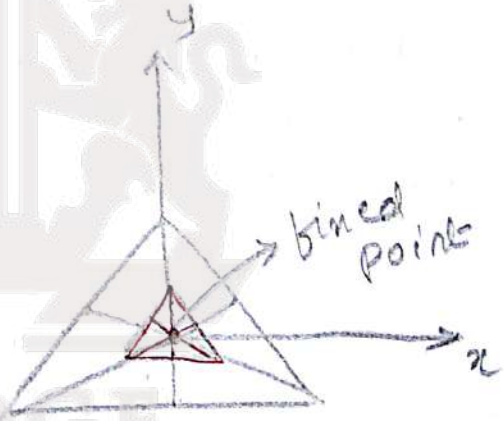
fixed point (x_f, y_f) – centroid

objects are now ~~rotated~~ resized by scaling the distances b/w object points & fixed points.

$$x' \cancel{=} x_f = (x - x_f) S_x, \quad y' - y_f = (y - y_f) S_y$$

$$x' = x \cdot S_x + x_f (1 - S_x)$$

$$y' = y \cdot S_y + y_f (1 - S_y)$$



Matrix Representations and Homogeneous co-ordinates

Three Basic two-Dimensional transformations

① Translation
② Rotation
③ Scaling

$$\boxed{P' = M_1 \cdot P + M_2 .}$$

P' & P represents column vectors.

- Matrix M_1 → 2×2 array containing multiplicative factors.

M_2 → 2 element ~~array~~ column matrix containing translational terms. $\begin{bmatrix} x_t \\ y_t \end{bmatrix}$

For translation, M_1 is identity matrix.

$P' = P + T$ where $T = M_2$

For rotation and scaling, M_2 is { contains translational terms associated with pivot point or scaling fixed point.

$P' = P + T$, $P' = R \cdot T$

{ multiplicative or Translational term are to be combined into a single matrix form }

Homogenous co-ordinates

A standard technique to expand the matrix representation for a 2D co-ordinate ~~representation~~ (x, y) position to a three-element (column matrix) representation (x_h, y_h, h) → called Homogeneous co-ordinates,

h → homogeneous parameter h (non zero value)

i.e (x, y) is converted into new co-ordinate values as (x_h, y_h, h)

$x_h = x \cdot h$
$y_h = y \cdot h$

$x = \frac{x_h}{h}$ $\quad y = \frac{y_h}{h}$

$\Rightarrow (x \cdot h, y \cdot h, h)$

Suppose $x=2$, $y=3$ & if $h=1$

$(x_h, y_h, h) = (2, 3, 1)$

If $h=2$. $(x_h, y_h, h) = (x.h, y.h, h)$

$= (2\times 2, 3\times 2, 2)$

$= (\underline{4}, \underline{6}, \underline{2})$

Again if you want to convert from homogenous co-ordinates to 2D then

$$x = \frac{x_h}{h} \quad \& \quad y = \frac{y_h}{h}$$

for the case $h=2$.

$x = \frac{4}{2} = 2$ $y = \frac{6}{2} = 3$. h is considered as '1'. i.e $h = \underline{1}$.

$(x, y) = (2, 3)$

If $h=1$, the old & the new co-ordinate values will not change

If $h=0$, then the co-ordinate system will be set to infinity.

Two-Dimensional Translation Matrix.

Using homogenous approach,

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

This translation operation can be written as

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

$T(t_x, t_y)$ is 3×3 translation matrix.

Two Dimensional Rotation Matrix :

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

(OR)

$$P' = R(\theta) \cdot P$$

Two Dimensional scaling Matrix .

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

(OR)

$$P' = S(S_x, S_y) \cdot P$$

Inverse Transformations

- For translation, we obtain the inverse matrix by negating the translation distances .

  Inverse translation Matrix is

$$T^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

- An Inverse Rotation is accomplished by replacing the rotation angle by its negative .

$$R^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & \theta \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ equavalent to its transpose.

$$R^{-1} = R^T$$

- Inverse matrix for scaling transformation. – by replacing the scaling parameters with their reciprocals

$$S^{-1} = \begin{bmatrix} \frac{1}{S_x} & 0 & 0 \\ 0 & \frac{1}{S_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- The inverse matrix generates an opposite scaling transformation, So any scaling matrix multiply X its Inverse produces Identity matrix.

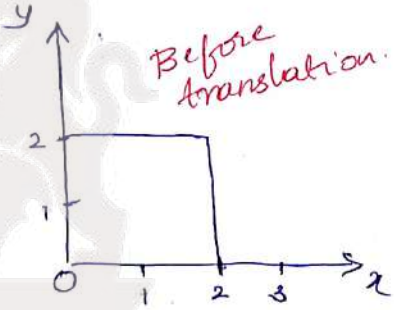
Translation Example.

1. Consider the square with co-ordinates (0,0) (2,0) (0,2) (2,2)

Translate square '2' units w.r.t. x-axis, '3' units with respect to 'y' axis.

$t_x = 2$, $t_y = 3$

translate each point w.r.t t_x & t_y.



① point (0,0)

$x=0$, $y=0$

$x' = x + t_x = 0+2 = 2$

$y' = y + t_y = 0+3 = 3$

(0,0) is translated to new point (2,3).

② point (0,2)

$x=0$, $y=2$

$x = 0+2 = 2$

$y = 2+3 = 5$

$(0,2) \rightarrow (2,5)$

④ point (2,0)

$x=2$, $y=0$

$x = 2+2 = 4$

$y = 0+3 = 3$

$(2,0) \rightarrow (4,3)$

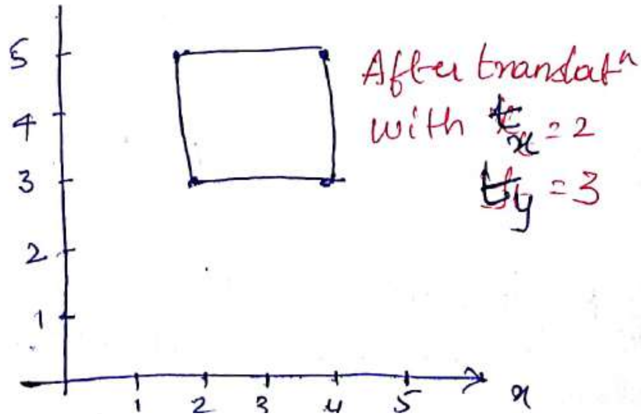
③ point (2,2)

$x=2$, $y=2$

$x = 2+2 = 4$

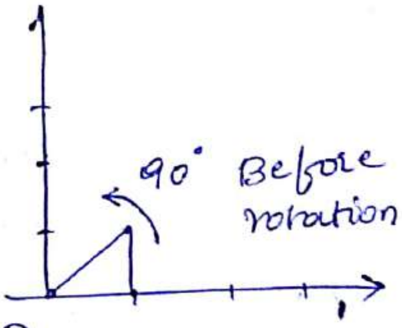
$y = 2+3 = 5$

$(2,2) \rightarrow (4,5)$



Rotation Example

1\. a) Consider a triangle (0,0) (1,0) & (1,1). Rotate 90° with Anticlockwise.



Soln :- formula for Anticlockwise.

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

We know that $\sin 90° = 1$ $\cos 90 = 0$

① Consider 1st co-ordinate

$(0,0) \Rightarrow x=0, y=0$

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

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$ $\Rightarrow (0,0) \rightarrow (0,0)$ does not change even after rotation

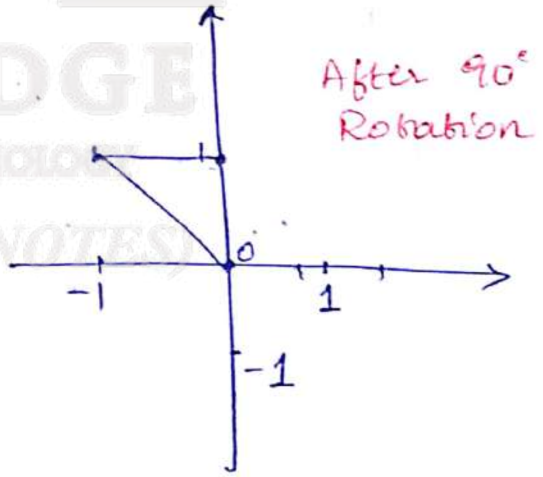
② 2nd co-ordinate

$(1,0) \Rightarrow x=1, y=0$

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

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

$(1,0) \rightarrow (0,1)$



③ 3rd co-ordinate

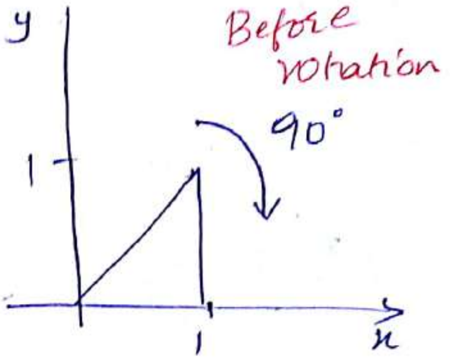
$(1,1) \Rightarrow x=1, y=1$

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

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

$(1,1) \rightarrow (-1,1)$

b) Rotating 90° in clockwise direction
triangle (0,0) (1,0)(1,1)



Before rotation

Sol^n

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

1) (0,0) - 1st co-ordinate   x=0, y=0

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

(0,0) → (0,0)

② 2nd co-ordinate (1,0)
x=1, y=0

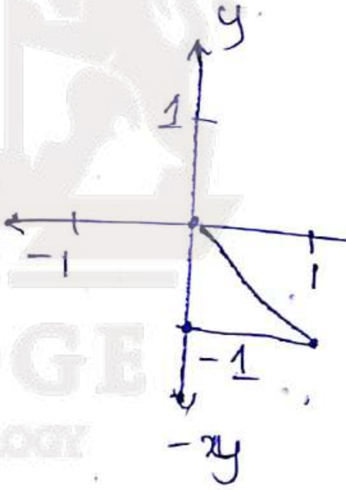
$$\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}0 & 1\\ -1 & 0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\-1\end{bmatrix}$$

(1,0) → (0,-1)

③ 3rd co-ordinate (1,1)
x=1, y=1

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

(1,1) = (1,-1)



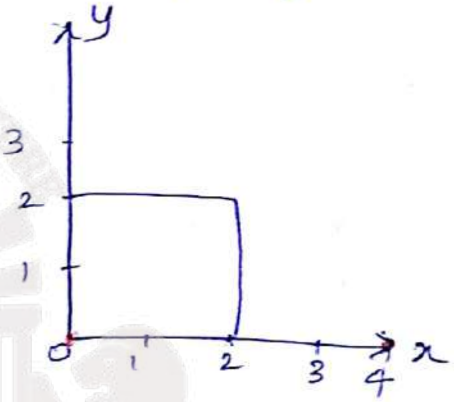
After rotation in clockwise 90°

Enample of Scaling

1. a) consider a square with co-ordinates $(0,0)$ $(2,0)$ $(0,2)$ $(2,2)$ Scale w.r.t. $S_x = 2$, $S_y = 3$

Soln: Since S_x & S_y > 0 → the size of the square increases

also since $S_x \neq S_y$ → hence the shape of the square also changes.



formula :- $x' = x * S_x$

$y' = y * S_y$

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

a. ① 1st cordinate $(0,0)$

$x = 0, y = 0$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$x' = x . S_x$

$= 0 . 2 = 0$

$y' = 0 . 3 = 0$

$(0,0) \rightarrow (0,0)$

② 2nd co-ordinate $(2,0)$

$x = 2$, $y = 0$

$x' = 2 * 2 = 4$

$y' = 3 * 0 = 0$

$(2,0) \rightarrow (4,0)$

③ 3rd co-ordinate $(0,2)$

$x = 0, y = 2$

$x' = 0 * 2 = 0$

$y' = 3 * 2 = 6$

$(0,2) \rightarrow (0,6)$

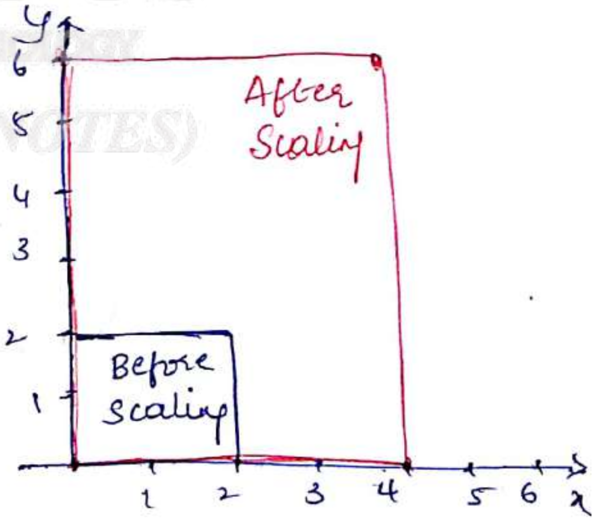
④ 4th co-ordinate $(2,2)$

$x = 2, y = 2$

$x' = 2 * 2 = 4$

$y' = 2 * 3 = 6$

$(2,2) \rightarrow (4,6)$



b) Scaling parameter $S_x = 0.5$, $S_y = 0.5$ for the same square (0,0) (2,0) (0,2) (2,2)

Sol'n. ① 1st co-ordinate

$x = 0, y = 0$

$x' = 0.5 \times 0 = 0$

$y' = 0 \times 0.5 = 0$

~~x~~ $(0,0) \rightarrow (0,0)$

② 2nd co-ordinate

$x = 2, y = 0$

$x' = 2 \times 0.5 = 1$

$y' = 0 \times 0.5 = 0$

$(2,0) \rightarrow (1,0)$

③ 3rd co-ordinate

$(0,2) \rightarrow x = 0, y = 2$

$x' = 0 \times 0.5 = 0$

$y = 2 \times 0.5 = 1$

$(0,2) \rightarrow (0,1)$

④ 4th co-ordinate

$(2,2) \rightarrow x = 2; y = 2$

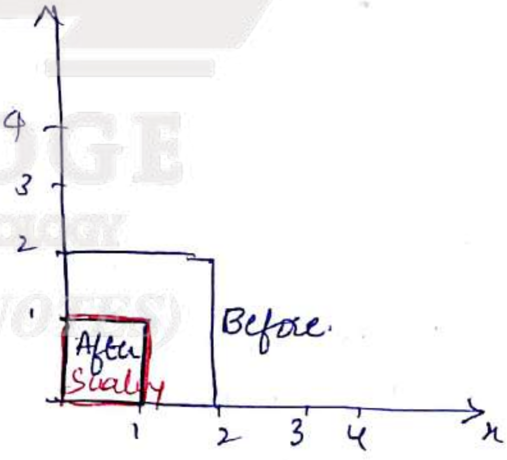
$x' = 2 \times 0.5 = 1$

$y' = 2 \times 0.5 = 1$

$(2,2) \rightarrow (1,1)$

Here S_x & S_y are of same values, hence the size of the object changes uniformly.

Also, S_x & S_y is in b/w 0 & 1, so the size will decrease.



Two - Dimensional composite Transformations.

* we can set up a sequence of transformations as a composite transformation matrix by calculating the product of individual transformation refered to as concatenation or composition

* If we want to apply two transformations to point position P,

$$P' = M_2 . M_1 . P$$
$$= M . P \quad \Rightarrow M = M_2 . M_1$$

Composite two-dimensional translations

* If two successive ~~transform~~ translation vectors (t_{1x}, t_{1y}) & (t_{2x}, t_{2y}) are applied to 2 dimensional co-ordinate position P, the final transformed location P' is calculated as,

$$P' = T(t_{2x}, t_{2y}) . \{T(t_{1x}, t_{1y}) . P\}$$
$$= \{T(t_{2x}, t_{2y}) . T(t_{1x}, t_{1y})\} . P$$

$$T(t_{2x}, t_{2y}) . T(t_{1x}, t_{1y}) = \begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & t_{1x}+t_{2x} \\ 0 & 1 & t_{1y}+t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= T(t_{1x}+t_{2x}, t_{1y}+t_{2y})$$

which dimonstrates 2 successive translations are additive.

Composite 2D Rotations

* Two successive rotation's applied to a point P produce the transformed position

$$P' = R(\theta_2).\{R(\theta_1).P\}$$
$$= \{R(\theta_2).R(\theta_1)\}.P$$

$$R(\theta_2).R(\theta_1) = R(\theta_1+\theta_2)$$

Composite 2D Scalings

Concatenation of transformation matrices for 2 successive scaling operations produces.

$$\begin{bmatrix} s_{2x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{1x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{1x}*s_{2x} & 0 & 0 \\ 0 & s_{1y}*s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(OR)

$$S(s_{2x}, s_{2y}).S(s_{1x}, s_{1y}) = S(s_{1x}.s_{2x}, s_{1y}.s_{2y})$$

General two-Dimensional Pivot-Point Rotation.

* when a graphics package provides only a rotate function w.r.t co-ordinate origin, we can generate a rotation about any pivot-point (x_r, y_r) by performing the sequence of translate - rotate - translate operations.

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

$$\begin{bmatrix}1&0&x_r\\0&1&y_r\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}\cos\theta&-\sin\theta&0\\\sin\theta&\cos\theta&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&0&-x_r\\0&1&-y_r\\0&0&1\end{bmatrix}$$

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

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

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

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

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

which can be expressed as

$$T(x_r\, y_r)\,.\,R(\theta)\,.\,T(-x_r,-y_r)=R(x_r,y_r,\theta)$$

where $T(-x_r,-y_r)=T^{-1}(x_r\, y_r)$

General Two-Dimensional Fixed point scaling

This sequence is

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

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

(OR)

$$T(x_f, y_f) \cdot S(s_x, s_y) \cdot T(-x_f, -y_f)$$

$$= S(x_f, y_f, s_x, s_y)$$

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

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

General Two-Dimensional Scaling Directions

Parameters s_x & s_y scale objects along the x & y directions. we can scale an object in other directions by rotating the object to allign the desired scaling directions.

To accomplish scaling without changing effecting the orientation, we first perform the rotation so that the direction for s_1 & s_2 co-inside with x & y axis, then scaling transformation $S(s_1, s_2)$ is applied.

$$R^{-1}(\theta) \cdot S(s_1, s_2) R(\theta) = \begin{bmatrix} s_1\cos^2\theta + s_2\sin^2\theta & (s_2-s_1)\cos\theta\sin\theta & 0 \\ (s_2-s_1)\cos\theta\sin\theta & s_1\sin^2\theta + s_2\cos^2\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

matrix concatenation properties

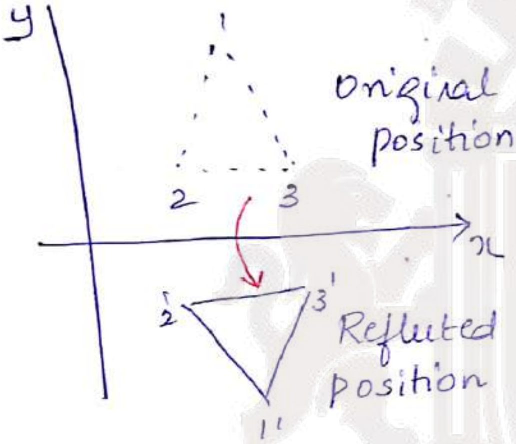
Associative : $M_3 \cdot M_2 \cdot M_1 = (M_3 \cdot M_2) \cdot M_1 = M_3 \cdot (M_2 \cdot M_1)$

Other 2D transformations.

- * Reflection
- * Shear

Reflection: A transformation that produces a mirror image of an object is called Reflection.

* Image is generated relative to an axis of reflection by rotating the object 180° about the reflection axis.

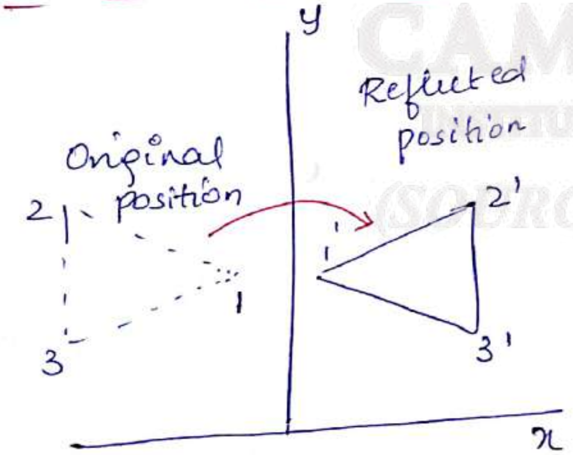


Reflection of an object about the x-axis.

Reflection about y=0

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

transformation retains remains x values, but "flips" the y values of co-ordinate posit^n.

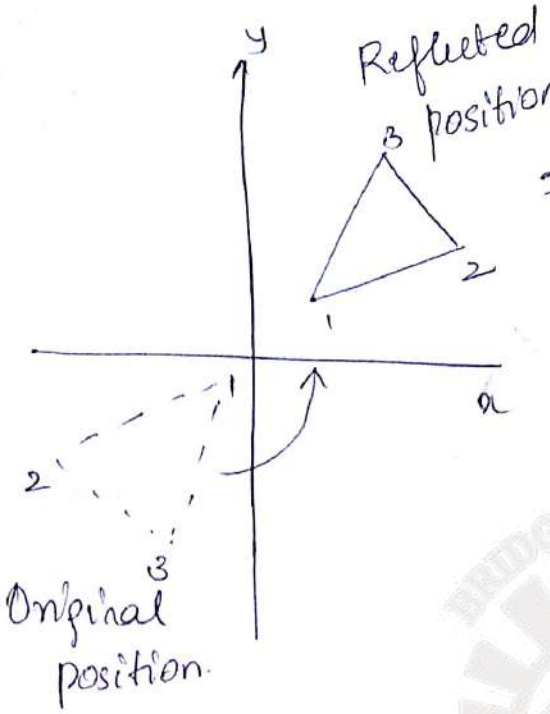


Reflection of an object about the y-axis.

A reflection about x=0 flips x co-ordinate while keeping y co-ordinate the same.

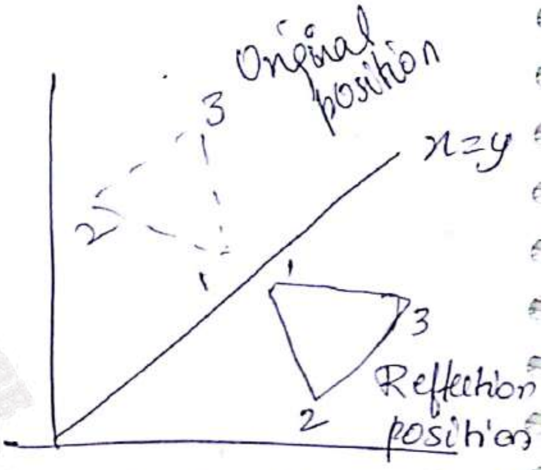
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

equivalent rotation = 18



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

Reflection about the Origin
(rotate about the xy plane)
180° $R(\theta)=180°$



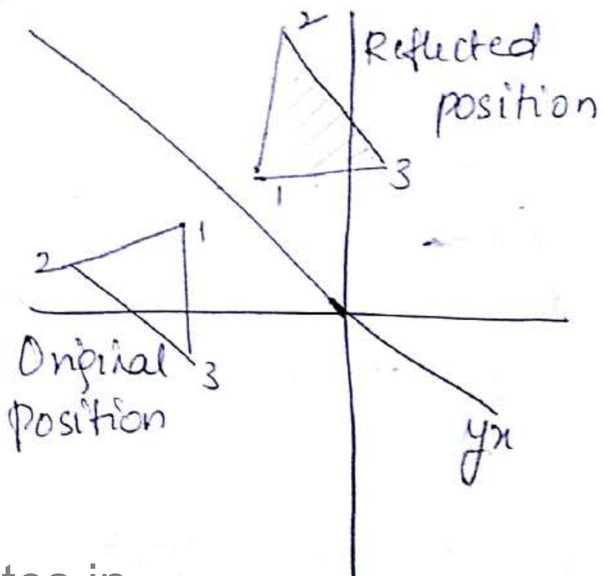
Reflection of an object with respect to line ~~x=~~ y = x

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

To obtain a transformation matrix for reflection about the diagonal $y=-x$, sequence is

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

① clockwise rotation by 45°
2) Reflection about y-axis (~~y=x~~)
3) counterclockwise rotation by 45°



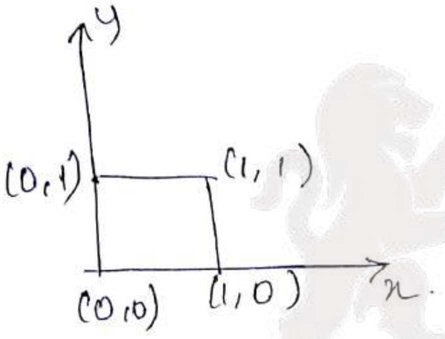
Shear

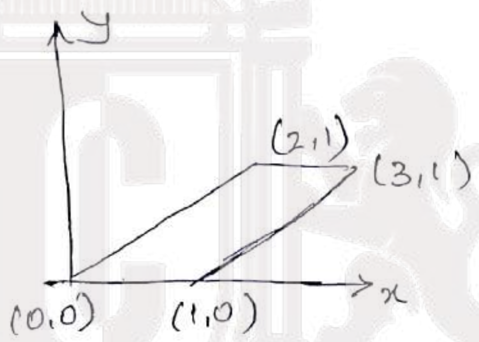
A transformation that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called shear.

An x-direction shear relative to x-axis is

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \boxed{\begin{array}{l} x = x + sh_x \cdot y \\ y = y \end{array}}$$

Sh_x → Shear parameter





Using x-direction shear with $sh_x = 2$.

① 1st co-ordinate $x=0, y=0$

$x = 0+0 = 0$

$y = 0$ (0,0) → (0,0)

② 2nd co-ordinate $x=1, y=0$

$x = 1+0 = 1$

$y = 0$ (1,0) → (1,0)

③ 3rd co-ordinate

$x=1, y=1$

$x = (1*2)+1 = 3$

$y = 1$

(1,1) → (3,1)

④ $x=0, y=1$

$x = 1*2 = 2$

$y = 1$ (0,1) → (2,1)

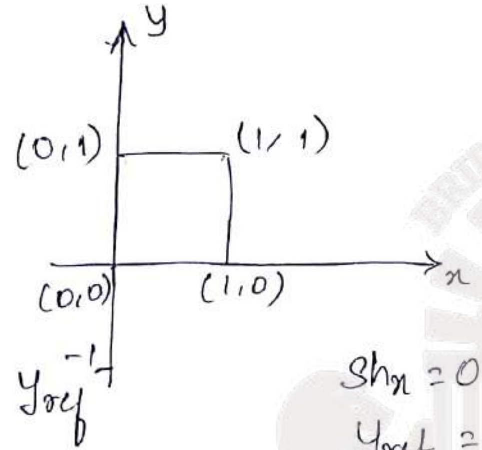
We can generate x-direction shears relative to other reference lines with $\{y_{ref} = -1\}$

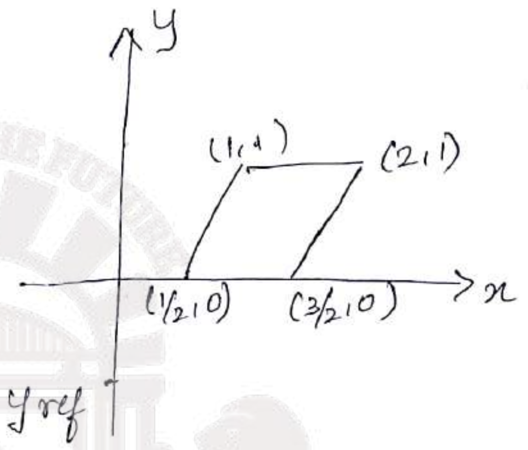
$$\begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \boxed{\begin{array}{l} x = x + sh_x(y - y_{ref}) \\ y = y \end{array}}$$

A y-direction shear relative to line $x = x_{ref}$ is generated with the transformation matrix.

$$\begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & -Sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boxed{\begin{aligned} x &= x, \\ y &= y + sh_y(x - x_{ref}) \end{aligned}}$$

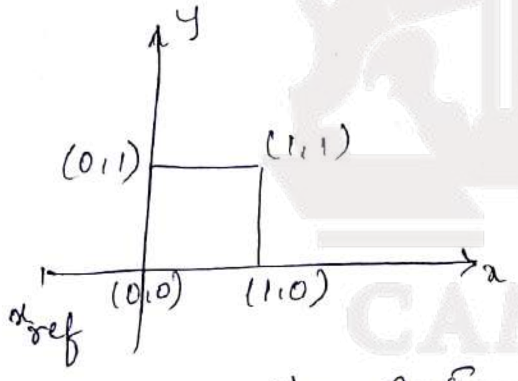


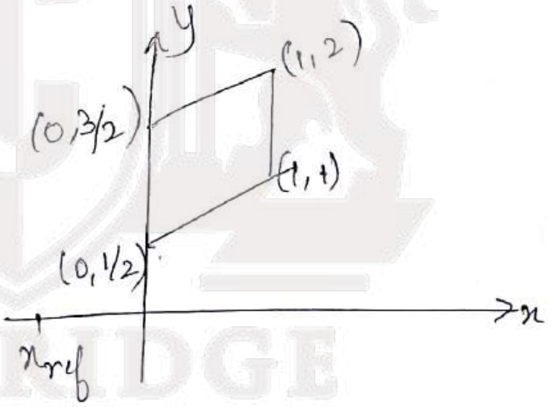


$Sh_x = 0.5$

$y_{ref} = -1$

in x-direction





$sh_y = 0.5$

$x_{ref} = -1$ in y direction

Rigid body transformation matrix.

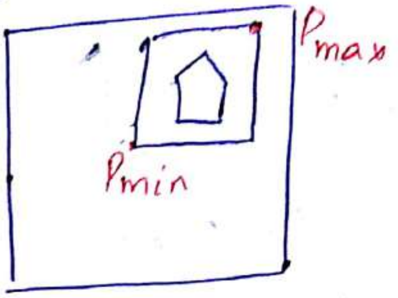
$$\begin{bmatrix} r_{xx} & r_{xy} & tr_x \\ r_{yx} & r_{yy} & tr_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$r_{xx}^2 + r_{xy}^2 = r_{yx}^2 + r_{yy}^2 = 1$$

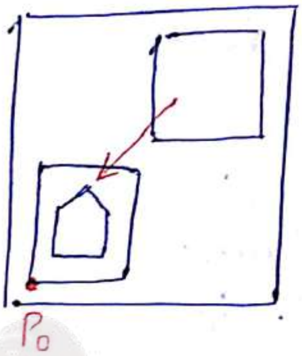
$$T(t_x, t_y) . R(x_r, y_r, \theta) = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r\sin\theta + t_x \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r\sin\theta + t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Raster Methods for Geometric Transformations

- Raster systems store picture information as color patterns in the frame buffer.



Functions that manipulate rectangular pixel arrays are called raster operations and moving a block of pixel values from one position to another is termed as block transfer, a bitblt or a pixblt



* Rotate a two-dimensional object or pattern 90° counterclockwise by reversing the pixel values in each row of the array, then interchanging rows & columns.

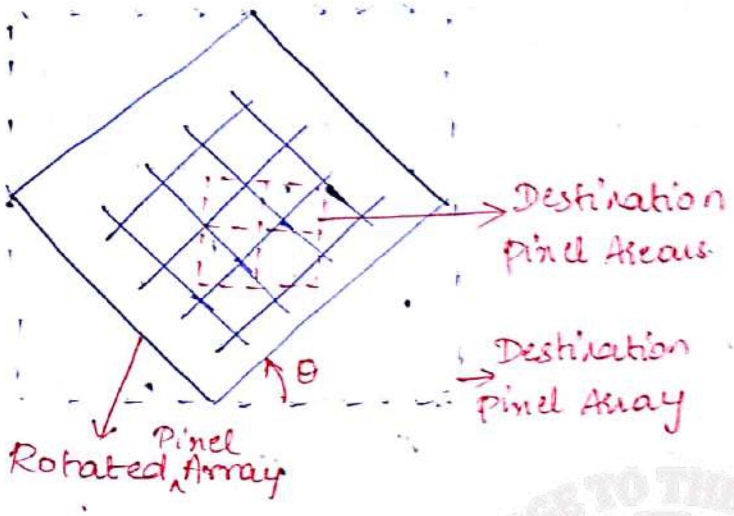
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \xrightarrow[\text{anti clock}]{\text{Reverse } 90^\circ} \begin{bmatrix} 3 & 6 & 9 & 12 \\ 2 & 5 & 8 & 11 \\ 1 & 4 & 7 & 10 \end{bmatrix} \xrightarrow[90^\circ]{\text{Reverse}} \begin{bmatrix} 12 & 11 & 10 \\ 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

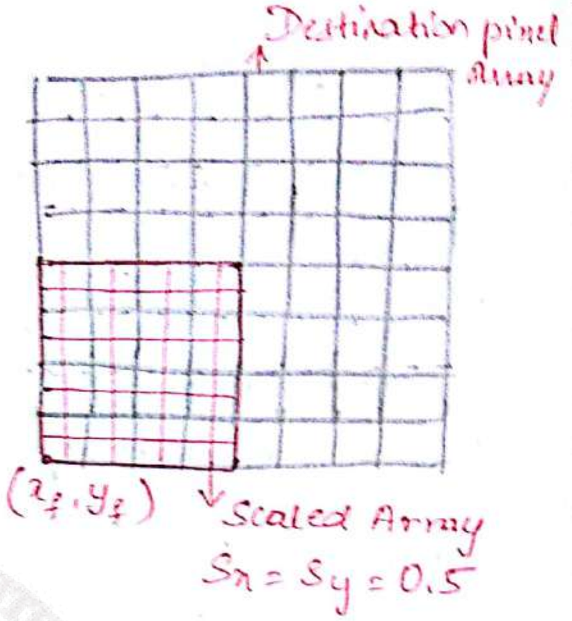
(Overall (180°) 90°+90°

* 180° rotation is obtained by reversing the order of elements in each row of the array, then reversing the order of rows.

* For rotations not multiples of 90°, additional processing is required.

Each destination pixel area is mapped onto the rotated array & amount of overlap with the rotated pixel area is calculated.





OpenGL Raster Transformations

- A translation of a rectangular array of pixel-color values from one buffer area to another by

$$\text{glCopyPixels}(\underbrace{xmin, ymin, width, height}_{\text{locations \& dimensions of pixel block}}, GL\_COLOR)$$

Specified that it is the color value that are to be copied.

- Both the regions to be copied (the source) & destination area should lie completely within the bounds of screen co-ordinates

- We can rotate a block 90° by first saving the block in an array, then rearranging the elements of the array & placing it back in refresh buffer.

Saving → glReadPixels (xmin, ymin, width, height, GL-RGB, GL_UNSIGNED_BYTE, colorArray);

placing back → glDrawPixels (width, height, GL-RGB, GL_UNSIGNED_BYTE, colorArray);

Two Dimensional scaling transformation

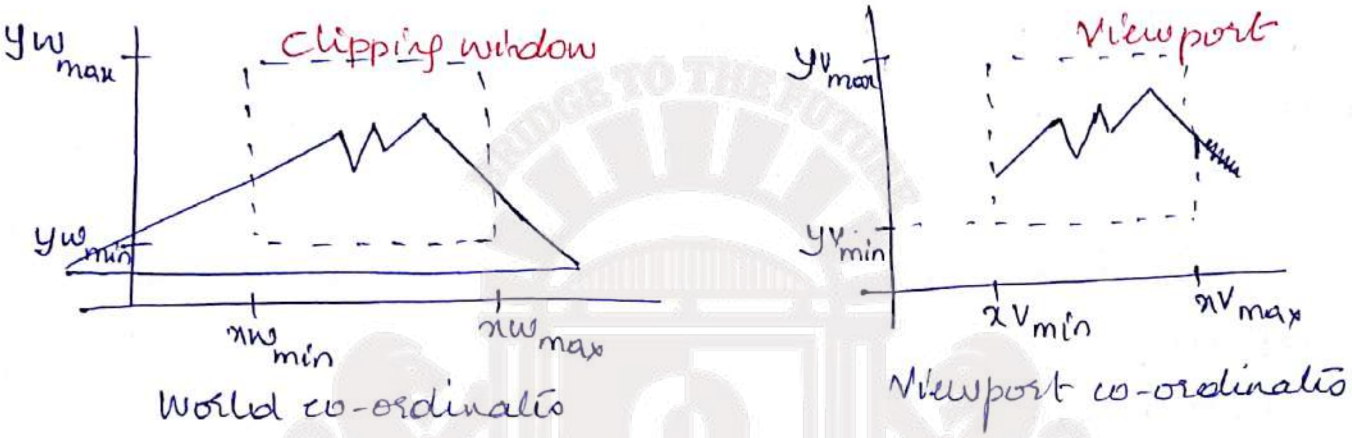
glPixelZoom(sx, sy) - scaling factors & then involving gl~~Read~~copyPixels ~~&~~ glDrawPixels

2D Viewing Pipeline.

A section of 2D scene that is selected for display is called clipping window.

Clipping window selects what we want to see.

Viewport indicates where it is to be viewed on the output devices.



- By changing positions of a viewport, objects can be viewed at different positions on display area.
- Multiple viewport windows can be used to display different ~~solid~~ sections of scene at different screen positions.
- In most applications, we tend to specify or define the objects with a convenient size, orientation, and location ~~with~~ in a seperate co-ordinate reference frame is called model or object frame.
- Each object must be brought into an application that might contain hundreds or thousands of individual objects. Constructing a scene by placing all the objects into appropriate locations within a scene reference frame is called world frame and the values are world co-ordinates.

- This step involves a transformation of individual object (modeling coordinates frames) to specified ~~to~~ position and orientations within the world frame.

- The mapping of 2D, world coordinate scene description to device co-ordinates is called a 2D viewing transformation [window to viewport transformation or windowing transformation]

- In 2D; clipping window is often just defined in world co-ordinates (viewing co-ordi.. are same as world co-ord)
- In 3D: A seperate viewing frame is required to specify the parameter for viewing position, direction and orientation.

- Transform viewing co-ordinates into Normalized co-ordinates where each co-ordinate value is in the range from 0 to 1, or -1 to 1.

- At the final → Map Normalized co-ordinates to Device co-ordinates., the contents of viewport are transferred to positions within the display window.

Refer figure 6-3 from textbook.

Open GL 2D Viewing functions.

Refer text book 6-4

Geometric Transformations in 3-D space.

2D rotation - rotations about axes that were perpendicular to xy plane.

3D rotation - select any spatial orientation for rotation axis.

3D Translation → how much the object is to be moved in each of the three co-ordinate direction.

3D Scaling → scale an object by choosing a scaling factor for each of three cartesian coordinates.

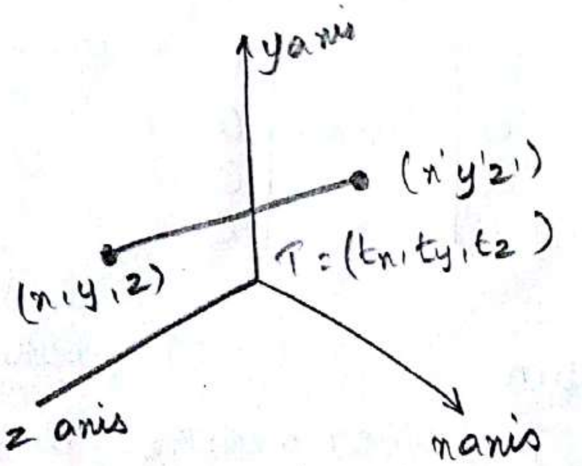
Three Dimensional Translation.

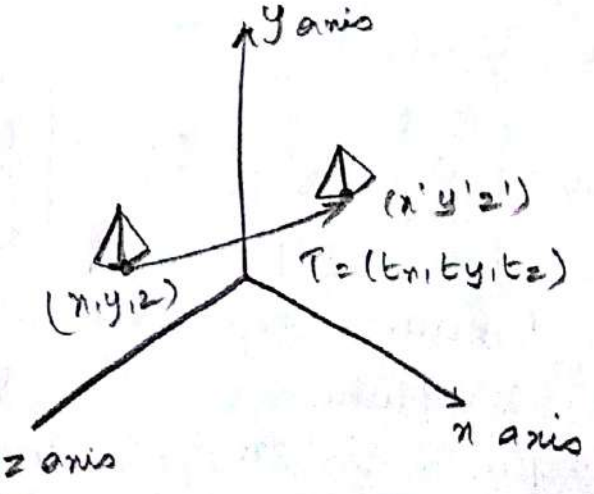
A position $P = (x, y, z)$ in 3D space is translated to a location $P' = (x', y', z')$ by adding translation distances t_x, t_y & t_z.

$$x' = x + t_x \quad y' = y + t_y \quad z' = z + t_z .$$

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

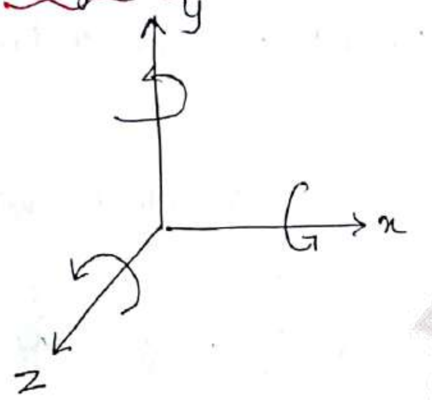
$$P' = T.P$$



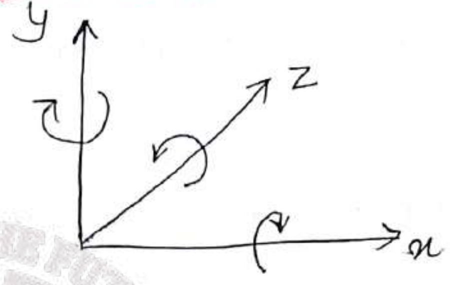


3D → depth of the object that we are viewing.
↳ projection geometry (object taken away becomes small
object bringing closer becomes zoomed to viewing eye)

Right handed space



Left handed space



Translation matrix in 3D

$$A = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & i & j & r \\ l & m & n & s \end{bmatrix} = \begin{bmatrix} T & K \\ V & \theta \end{bmatrix}$$

$$T = \begin{bmatrix} a & b & c \\ d & e & f \\ g & i & j \end{bmatrix}$$ produce linear transformation: Scaling, shearing, reflection & rotation

$K = [p\ q\ r]^T$, produces translation

$V = [l\ m\ n]^T$, yields perspective transformation

s – responsible for uniform scaling in 3D

3D Reflection

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

Produces reflection about:
XY plane   YZ plane   ZX plane

A reflection in 3D space can be performed relative to a selected reflection axis or w.r.t reflection plane

Rotation Matrices along an axis.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

x - axis.

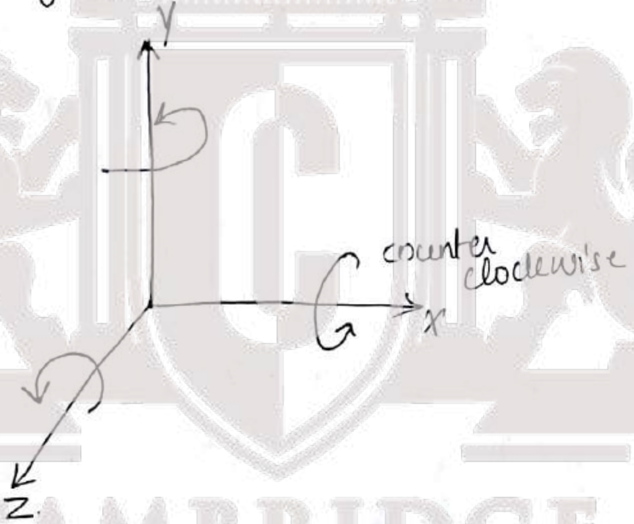
$$\begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

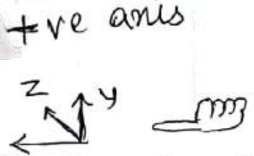
y - axis.

$$\begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

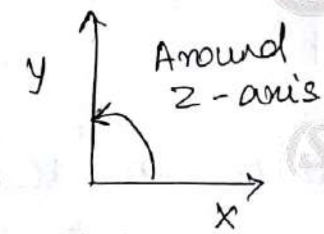
z - axis

why is the sign reversed in one case.

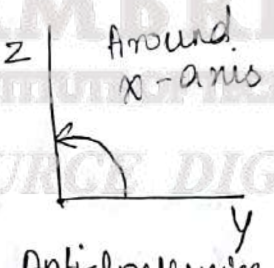




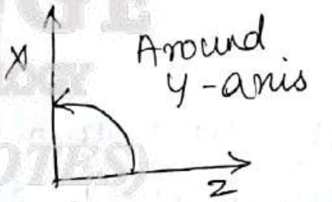
Thumb pointing to arrow from origin, direction is known from the direction of rotatn



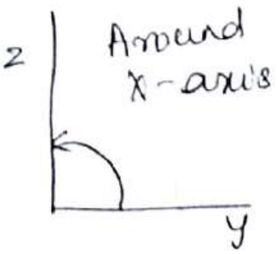
Anti clockwise



Anticlockwise



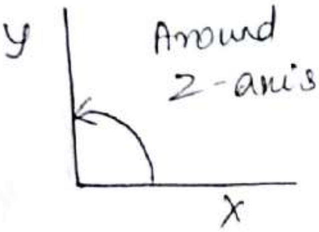
Anti clockwise



$$\begin{array}{c}\left[\begin{array}{cccc}1 & 0 & 0 & 0\\ 0 & \cos\alpha & -\sin\alpha & 0\\ 0 & \sin\alpha & \cos z & 0\\ 0 & 0 & 0 & 1\end{array}\right]\end{array}\begin{array}{l}\leftarrow X\\ \leftarrow Y\\ \leftarrow Z\\ \\ \end{array}$$

from y to z

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

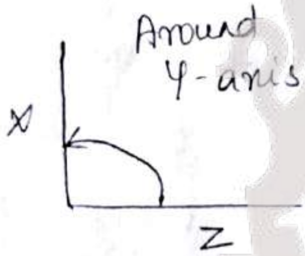


$$\left[\begin{array}{cccc}\cos\gamma & -\sin\gamma & 0 & 0\\ \sin\gamma & \cos\gamma & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{array}\right]\begin{array}{l}\leftarrow X\\ \leftarrow Y\\ \leftarrow Z\\ \\ \end{array}$$

from x to y

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

If -ve sign is here, the rotation would be from x to z



$$\left[\begin{array}{cccc}\cos\beta & 0 & \sin\beta & 0\\ 0 & 1 & 0 & 0\\ -\sin\beta & 0 & \cos\beta & 0\\ 0 & 0 & 0 & 1\end{array}\right]\begin{array}{l}\leftarrow X\\ \leftarrow Y\\ \leftarrow Z\\ \\ \end{array}$$

from z to x

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

generalising α, β, γ with θ

General 3D Rotation

An object is to be rotated about the axis that is ‖el to one of co-ordinate axis, sequence is

1) Translate the object so that the rotation axis co-insides with the ‖el co-ordinate axis
2) Perform the specified rotation about that axis.
3) Translate the object so that the rotation axis is moved back to its original position.

$$P' = T^{-1}.R_x(\theta).T.P$$
$$R(\theta) = T^{-1}.R_x(\theta).T$$

General 3D Rotations
Rotation about an Arbitrary Axis in space.

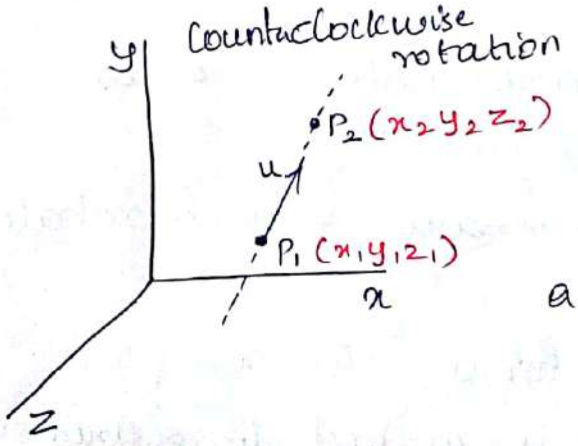
Assume we want to perform a rotation by θ degrees, about an axis in space passing through the point (x_0 y_0 z_0) with direction cosines (c_x, c_y, c_z)

1. ~~First~~ translate ~~by~~ the object so that ~~the~~ rotation axis co-incides with the parallel co-ordinate ~~axis~~.

~~2.~~ .

1. Translate the object so that the rotation axis passes through the co-ordinate origin.
2. Rotate the object so that the axis of rotation co-incides with one of the co-ordinate axis.
3. Perform the specified rotation about the selected co-ordinate axis
4. Apply inverse rotations to bring the rotation axis back to its original position/orientation.
5. Apply the inverse translation to bring the rotation axis back to its original spatial position.

- transform the rotation axis onto any one of the three co-ordinate axis. The 'z' axis is often convienient choice



Direction of rotation is counter clockwise from P_2 to P_1

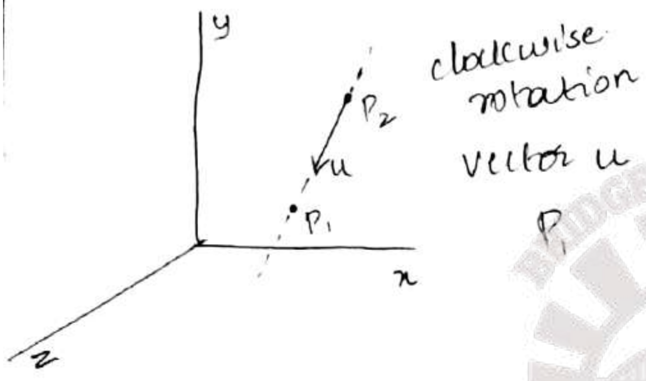
The components ~~of~~ of rotation axis vector are computed as

$$V = P_2 - P_1$$
$$= (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

unit rotation axis vector u is

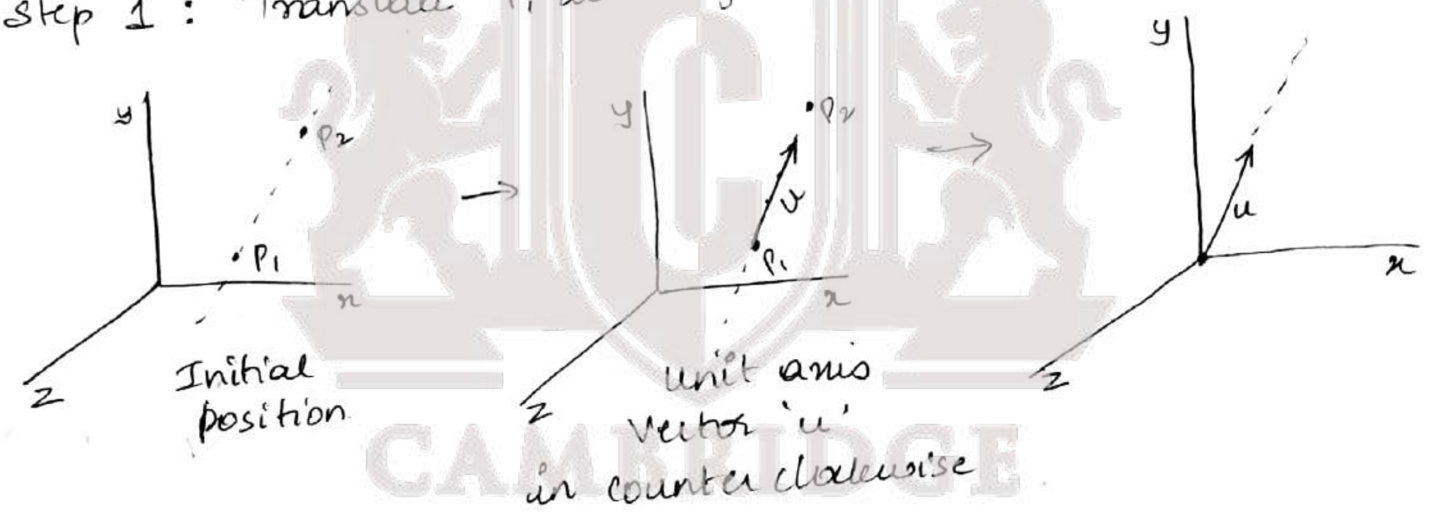
$$u = \frac{V}{|V|} = (a, b, c)$$

$$a = \frac{x_2 - x_1}{|V|}, \quad b = \frac{y_2 - y_1}{|V|}, \quad c = \frac{z_2 - z_1}{|V|}$$



vector u points in the direction from P_2 to P_1

Step 1 : Translate P_1 to origin



$$T = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

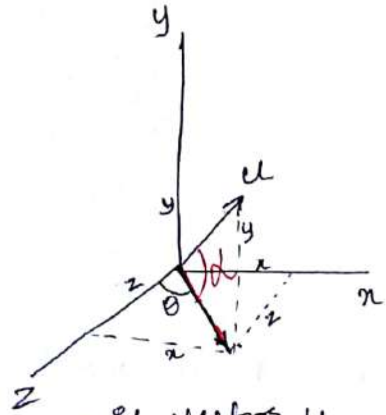
Step 2 : Place the rotation axis onto z axis

This can be accomplished by

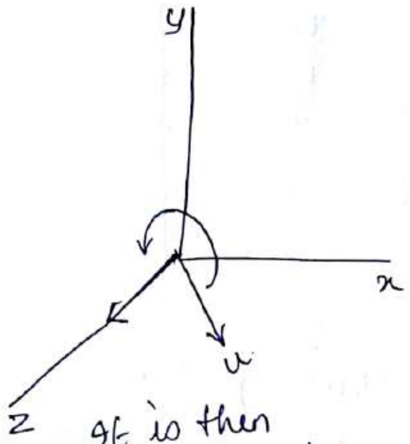
1) first rotate about the x-axis & then rotate about the y axis.

The x-axis rotation gets vector u into xz plane

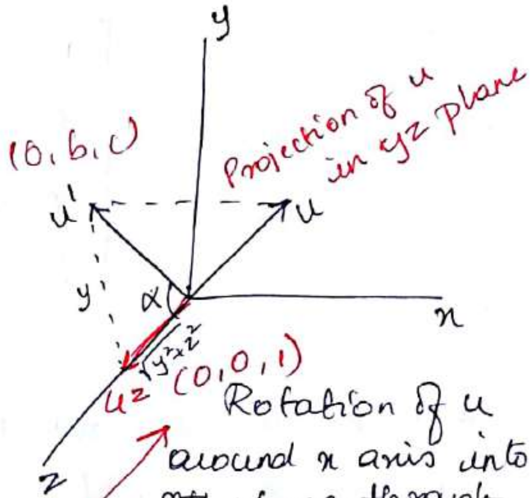
y-axis rotation swings u around to z axis



unit vector u is rotated about the x-axis - on xz plane ①



It is then rotated around y axis to allign it with z axis



Rotation of u around x axis into x~~z~~ plane through an angle α on z' axis

- Vector dot product → to determine cosine term.
- vector cross product → to calculate sine term
- Rotation angle α is the angle between the projection of u in yz plane & the positive z axis.
- projection of u in yz plane as the vector $u' = (0, b, c)$

i.e $\cos\alpha = \frac{u' \cdot u_z}{|u'||u_z|} = \frac{c}{d}$ $\left\{\frac{\text{adjacent}}{\text{hypo}}\right\}$

where $d = \sqrt{b^2+c^2}$ magnitude of u'

- determine the $\sin\alpha$ — cross product of u' & u_z

$$u' \times u_z = u_x |u'||u_z| . \sin\alpha$$

$$u' \times u_z = u_x . b \quad \text{substitute}$$

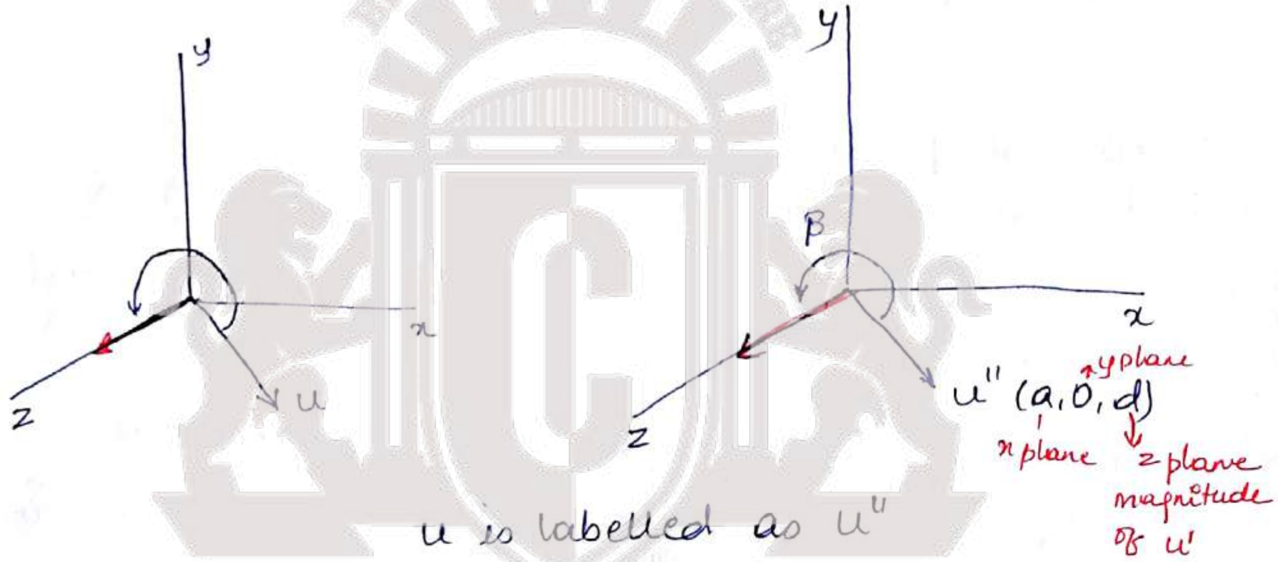
$$\cancel{u_x} . b = \cancel{u_x} . d . \sin\alpha$$

$$b = d . \sin\alpha$$

$$\sin\alpha = \frac{b}{d}$$

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & -\frac{b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

* Next is to rotate/swing the unit vector in xz plane ~~to~~ counter clockwise around y axis onto the positive z axis.



u is labelled as u''

rotation angle β. since $|u_z| = |u''| = 1$

$$\cos\beta = \frac{u''.u_z}{|u''|.|u_z|} = d$$

determining sin ~~α~~β

$$u'' \times u_z = u_y\, |u''|\, |u_z| \sin\beta$$

$$u'' \times u_z = u_y.(-a) \quad \text{subst}$$

hence, $\cancel{u_y}.(-a) \,.\, \cancel{u_y}.1.\sin\beta$

$$\sin\beta = -a.$$

$$R_y(B) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 3: Rotate the object with angle θ around the 'z' axis

$$R_2(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

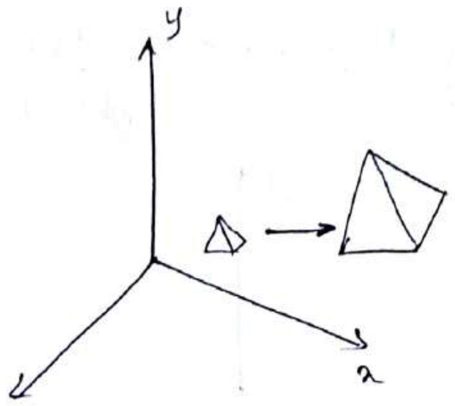
Step 4: Rotate the Axis to its Original Orientation.

Step 5: Apply inverse transform to translate the rotation axis to its original position

$$\therefore R(\theta) = T^{-1} R_x^{-1}(\alpha) R_y^{-1}(\beta) . R_z(\theta) . R_y(B) . R_x(\alpha) . T$$

Three Dimensional Scaling

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



Doubling the size of the object with z transformation also moves the object farther from the origin.

$P' = S.P$ $(S_x S_y S_z) \rightarrow$ scaling parameters

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

- * parameter value greater than 1 moves a point farther from the origin
- * parameter value less than 1 moves a point closer to the origin.
- * If scaling parameter are all not equal, relative dimensions of a transformed object are changed.

Scaling transformation w.r.t fixed point (x_f, y_f, z_f).

Sequence is

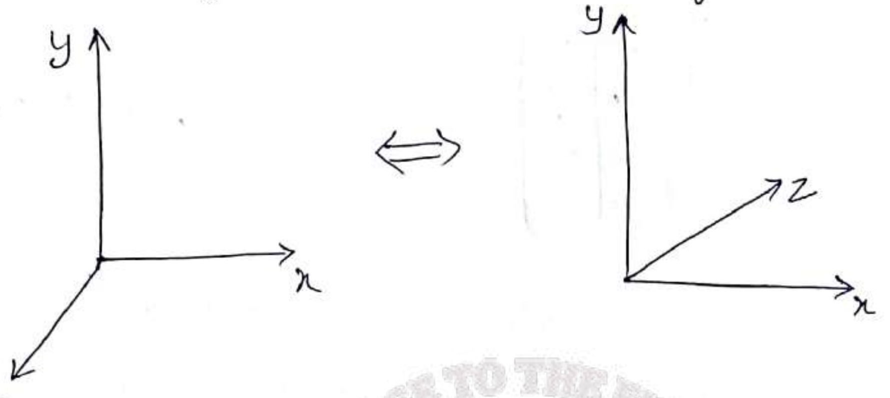
1) Translate the fixed point to origin.
2) Apply the scaling transformation relative to the co-ordinate origin.
3) Translate the fixed point back to its original position

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

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

3D reflection Continued.

Right handed → left handed



This transformation changes the sign of 'z' co-ordinates, leaving the values of x & y co-ordinates unchanged.

3D Shears

- used to modify object shapes.
- applied in 3D viewing transformation for perspective projection.
- shears relative to z-axis

$$M_{z\,shear} = \begin{bmatrix} 1 & 0 & sh_{zx} & -sh_{zx} \cdot z_{ref} \\ 0 & 1 & sh_{zy} & -sh_{zy} \cdot z_{ref} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Affine Transformations

A co-ordinate transformation of the form

$$x' = a_{xx}x + a_{xy}y + a_{xz}z + b_x$$
$$y' = a_{yx}x + a_{yy}y + a_{yz}z + b_y$$
$$z' = a_{zx}x + a_{zy}y + a_{zz}z + b_z$$

is called an affine transformation: Each of the transformed coordinates x', y' and z' is a linear function of the original co-ordinates x, y & z.

parameters a_{ij} & b_k are constants determined by the transformation type.

General properties of affine transformations

- parallel lines are transformed into parallel lines.
- finite points map to finite points.

Ex of Affine transformations: Translation, rotation, scaling, reflection & shear.

Basic OpenGL Geometric Transformations

- A 4x4 translation matrix is constructed with the following routine

  glTranslate* (tx, ty, tz);

  Ex: glTranslatef (25.0, -10.0, 0.0);

  translate 25 units in x direction and -10 units in y-direction.

- A 4x4 rotation matrix is generated with

  glRotate* (theta, vx, vy, vz);

vector (vx, vy, vz) defined the orientation for a rotation axis that passes through the co-ordinate origin.

glRotatef (90.0, 0.0, 0.0, 1.0);

sets up the matrix for a 90° rotation about the z-axis.

- A 4×4 scaling matrix.

glScalef (sx, sy, sz)

This function will also generate reflections when -ve values are assigned to scaling parameters.

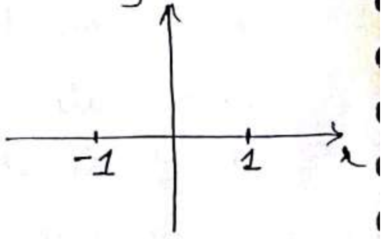
glScalef (2.0, -3.0, 1.0);

- It produces a matrix that scales by a factor of 2 in x direction, scales by a factor of 3 in y-direction, & reflects with respect to x axis.

** Quaternion Methods for 3-D Rotations

Quaternions are extensions of 2D complex numbers, more often important in animations that requires complicated motion sequences & motion interpolations between two given positions of an object.

Quaternions are ordered pair, consisting of a scalar part & vector part. $q = (s, v)$.



Basics of complex numbers : $a = x + iy$

$x \rightarrow$ real part, y = imaginary part,

$i = \sqrt{-1}$

Imaginary unit

A pure imaginary number with $y = 1$ is called imaginary unit & is denoted as $i = (0,1)$

$i^2 = (0,1)(0,1)$

product of two complex number

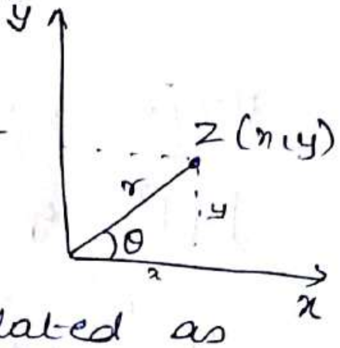
$(x_1,y_1)(x_2,y_2) = (x_1x_2 - y_1y_2,\ x_1y_2 - x_2y_1)$

$i^2 = (0,1)(0,1) = (-1,0)$

Therefore i^2 is a real number -1, Hence $i = \sqrt{-1}$

Polar-Cordinate representation of complex number.

$x = r\cos\theta$, $y = r\sin\theta$



$z = r(\cos\theta + i\sin\theta)$ where $r = \sqrt{x^2+y^2}$

i.e $x^2+y^2 = r^2\cos^2\theta + r^2\sin^2\theta$

$\boxed{x^2+y^2 = r^2}(\cos^2\theta + \sin^2\theta)$

Hence $r = \sqrt{x^2+y^2}$ & θ can be calculated as

$\frac{y}{x} = \tan\theta \Rightarrow \theta = \tan^{-1}(y/x)$

$z = r(\cos\theta + i\sin\theta)$ can be written as $z = re^{i\theta}$

where $e^{i\theta} = \cos\theta + i\sin\theta$

Hence complex numbers ~~can~~ are extended to higher dimensions using Quaternions.

$q = s + ia + jb + kc$ $\quad v = ia + jb + kc$

a, b, c - co-efficients in the imaginary part.

s - real part called scalar part.

$i^2 = j^2 k^2 = -1$, $ij = -ji = k$

$jk = -kj = i$

$ki = ik = j$

$q_1 + q_2 = (s_1 + s_2) + i(a_1 + a_2) + j(b_1 + b_2) + k(c_1 + c_2)$

Hence for Quaternion of form $q = (s, v)$

$q_1 + q_2 = (s_1 + s_2,\ v_1 + v_2)$

$q_1 \cdot q_2 = (s_1 s_2 - v_1 \cdot v_2,\ s_1 v_2 + s_2 v_1 + v_1 \times v_2)$

$(s_1, v_1) \cdot (s_2\ v_2)$

[illegible] $q = s + ai + bj + ck.$

$q = (s, v)$

$|q|^2 = q \cdot q = (s, a\,i + b\,j + c\,k) \cdot (s, a\,i + b\,j + c\,k)$

i.e $q \cdot q = (a\,i + b\,j + c\,k)(a\,i + b\,j + c\,k)$

Here $i.j = 0,\ j.k = 0,\ i.k = 0,\ i.i = -1,\ j.j = -1,$

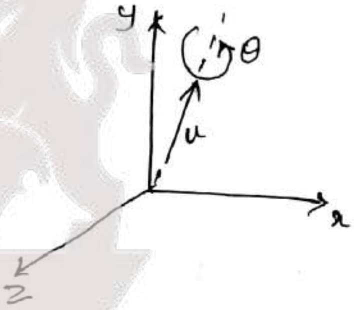
$k.k = -1,\ i.j.k = -1$

Hence $q \cdot q = a^2 + b^2 + c^2$

$\therefore |q|^2 = q \cdot q = s^2 + v.v$

$$q^{-1} = \frac{1}{|q|^2}(s, -v)$$

so that $q\,q^{-1} = q^{-1}\,q = (1, 0)$



$q = (s, v)$

where $s = \cos\frac{\theta}{2}$, $v = u \sin\frac{\theta}{2}$

any point position P that is rotated by this quaternion can be represented in quaternion notation as

$P = (0, p)$

'P' is Vector part, $p = (x, y, z)$. The rotation of point is then carried out with quaternion operation

$P' = q\,P\,q^{-1}$ 'q' is a quaternion operation, get it to origin rotate & replace it where $q^{-1} = (s, -v)$

This transformation produces a new quaternion

$P' = (0, p')$

where $p' = s^2 p + v(p.v) + 2s(v \times p) + v \times (v \times p)$

Hence the overall composite rotation matrix.

$R_x^{-1}(\alpha) \cdot R_y^{-1}(\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha)$ in a 3 by 3 form as

$$M_R(\theta) = \begin{bmatrix} 1-2b^2-2c^2 & 2ab-2sc & 2ac+2sb \\ 2ab+2sc & 1-2a^2-2c^2 & 2bc-2sa \\ 2ac-2sb & 2bc+2sa & 1-2a^2-2b^2 \end{bmatrix}$$

Use trignometrical identities to simplify the terms.

$$\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2} = 1 - 2\sin^2\frac{\theta}{2} = \cos\theta.$$

$$\& \quad 2\cos\frac{\theta}{2}\sin\frac{\theta}{2} = \sin\theta.$$

Using this rewrite the matrix.

$\therefore \; M_R(\theta) =$

$$\begin{bmatrix} u_x^2(1-\cos\theta)+\cos\theta & u_x u_y(1-\cos\theta)-u_z\sin\theta & u_x u_z(1-\cos\theta)+u_y\sin\theta \\ u_y u_x(1-\cos\theta)+u_z\sin\theta & u_y^2(1-\cos\theta)+\cos\theta & u_y u_z(1-\cos\theta)-u_x\sin\theta \\ u_z u_x(1-\cos\theta)-u_y\sin\theta & u_z u_y(1-\cos\theta)+u_x\sin\theta & u_z^2(1-\cos\theta)+\cos\theta \end{bmatrix}$$

A complete quaternion rotation expression.

$$R(\theta) = T^{-1} \cdot M_R \cdot T.$$

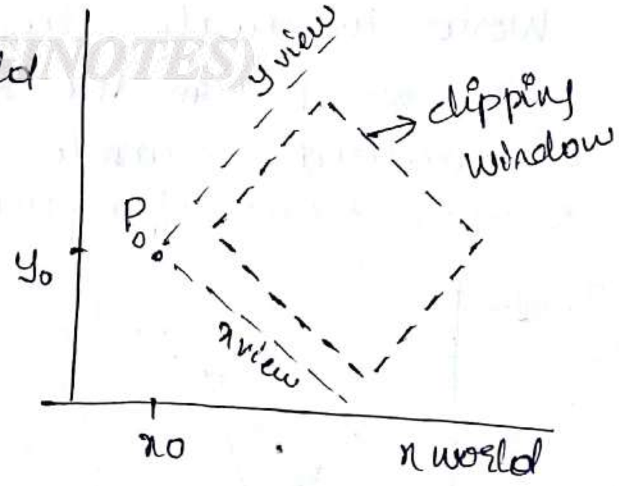
The clipping window

- clipping window can be designed with any shape, size & orientation as we choose.
- clipping with star patterns, an ellipse, or a figure with spline boundary requires more processing than clipping against a rectangle.
- It is hard to determine where the objects intersects a circle than to find out where it intersects a straight line.
- More often graphics packages commonly allow only rectangular clipping windows alligned with x & y axes.
- Rectangular clipping windows in standard position are easily defined by giving the co-ordinates of two opposite corners of each rectangle.

Viewing-coordinate clipping window

- 2D viewing transformation - is to set up a viewing coordinate system within the world-coordinate frame.
- It ~~provides~~ (viewing frame) provides a reference for specifying a rectangular clipping window with any selected orientation and position.

- We choose the origin for a 2D viewing co-ordinate frame at some world position $P_0 = (x_0, y_0)$ & establish the orientation using vector V that defines the y_{view} direction.



Vector V is called 2D view up vector.

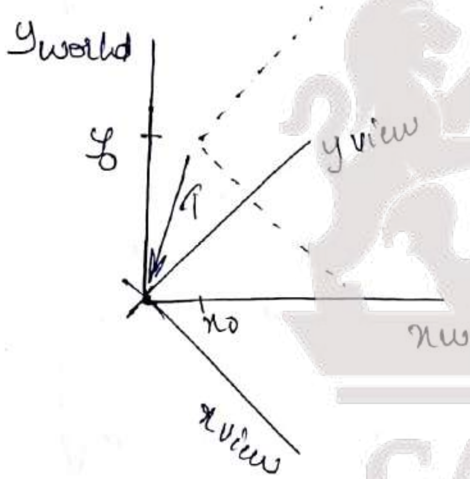
Super Impose viewing frame on world frame

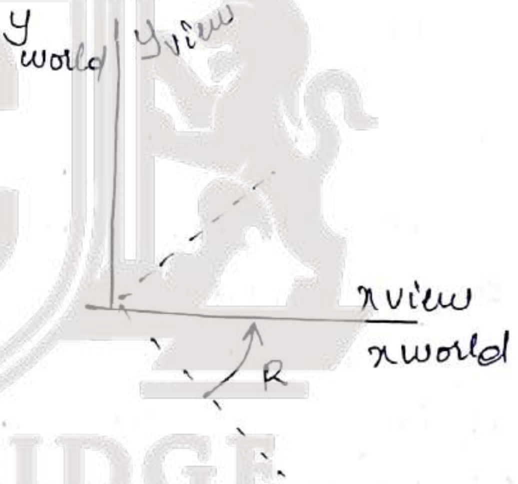
1. Translate viewing origin to the world origin.
2. Rotate the viewing system to allign it with the world frame.

Given the orientation vector V, we calculate the components of unit vectors $v = (v_x, v_y)$ $u = (u_x, u_y)$ for the yview & xview axes respectively.

Objects positions in world coordinates are then converted to viewing coordinates with composite 2D transformation matrix.

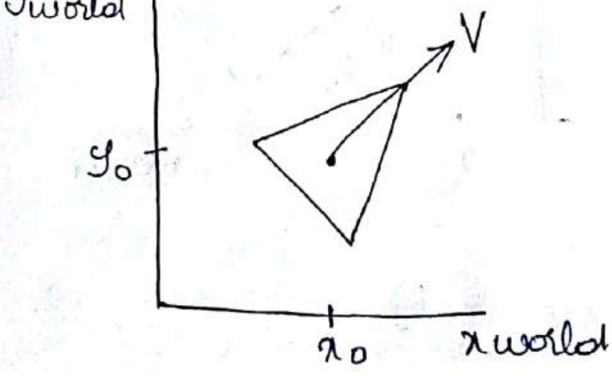
$$M_{wc,vc} = R.T$$

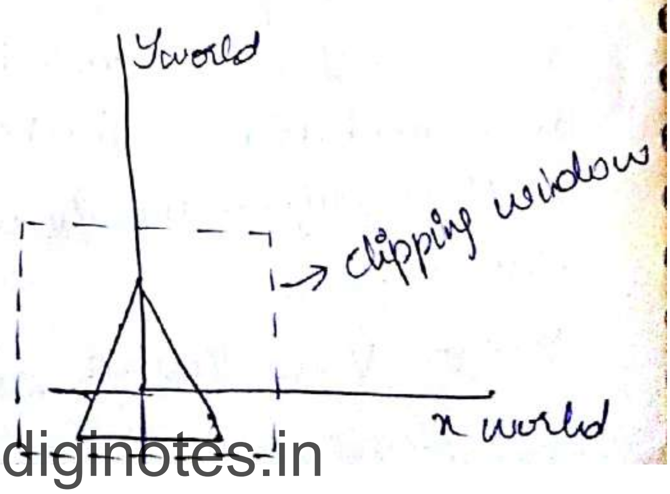




World coordinate clipping window

we perform the same above steps by considering a standard rectangle but without considering a viewing frame of reference





Normalization and Viewport transformation.

Viewport co-ordinatis are often given in range from 0 to 1., hence viewport is positioned with a unit square. After clipping, the unit vector containing viewport is mapped to o/p display device.

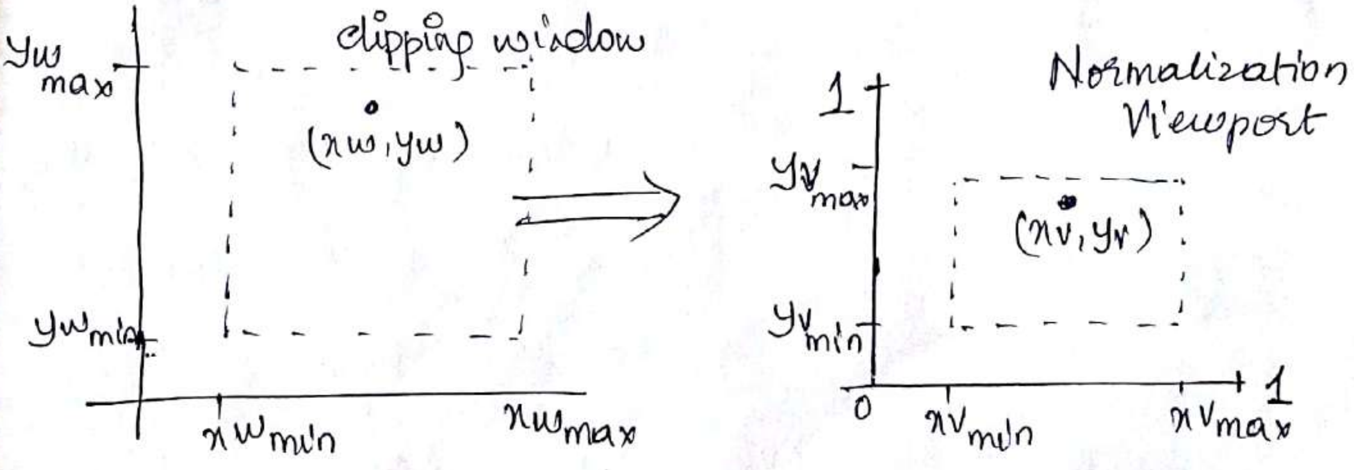
Mapping the clipping Window into a Normalized Viewport

- First a viewport is defined with normalized coordinates values between 0 and 1.
- Same relative placement of a point is maintained in the viewport as it had in clipping window i.e the relative position will not change.
- i.e based on the position we need to find relative viewport coordinates based on window coordinates.

Position (xw, yw) in the clipping window is mapped into position (xv, yv) in associated viewport.

$$\frac{x_v - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

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



Scaling parameters S_x & S_y are

$$S_x = \frac{XV_{max} - XV_{min}}{X_{wmax} - XW_{min}} \quad \& \quad S_y = \frac{YV_{max} - YV_{min}}{Y_{Wmax} - Y_{wmin}}.$$

{how many times the viewport is greater than window}

As we have all values other than X_v & Y_v, which is the coordinate in viewport, hence X_v.

$$X_v - X_{Vmin} = (XV_{max} - X_{Vmin}) \frac{(x_w - x_{wmin})}{(X_{Wmax} - X_{Wmin})}$$

$$= (x_w - x_{wmin}) \frac{(XV_{max} - XV_{min})}{(XW_{max} - XW_{min})}$$

Substitute S_x.

$$X_v - XV_{min} = (x_W - x_{Wmin}) S_x$$

$$\boxed{X_V = XV_{min} + (x_w - X_{wmin}) S_x}$$

|||ly

$$\boxed{y_v = Y_{vmin} + (y_w - y_{wmin}) S_y}$$

$$X_V = XV_{min} + S_x . x_w - S_x . X_{Wmin}$$

$$= S_x . x_w - S_x . X_{Wmin} + XV_{min} \quad \text{Substitute } S_x$$

$$= S_x . x_w - \frac{XV_{max} XW_{min} + XV_{min} XW_{min}}{XW_{max} - XW_{min}} + XV_{min}$$

multiply & take common.

$$= S_x . x_w - \frac{XV_{max} XW_{min} + \cancel{XV_{min} XW_{min}} + XV_{min} XW_{max} - \cancel{XV_{min} XW_{min}}}{XW_{max} - XW_{min}}$$

$$= S_x . x_w + \frac{XW_{max} X_{Vmin} - XV_{max} X_{wmin}}{XW_{max} - XW_{min}}$$

$$\boxed{x_v = s_x x_w + t_x}$$

where $t_x = \dfrac{xw_{max} xv_{min} - xv_{max} xw_{min}}{xw_{max} - xw_{min}}$

Also $y_v = yv_{min} + (y_w - yw_{min}) s_y$.

$$y_v = yv_{min} + s_y \cdot y_w - s_y \cdot yw_{min}.$$

$$= s_y \cdot y_w - \left(\frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}\right) \cdot yw_{min} + yv_{min}.$$

$$= s_y \cdot y_w - \frac{yw_{min} yv_{max} + yv_{min} yw_{min}}{yw_{max} - yw_{min}} + yv_{min}.$$

$$= s_y \cdot y_w - \frac{yw_{min} yv_{max} + \cancel{yv_{min}} yw_{min} + yv_{min} yw_{max} - \cancel{yv_{min}} . yw_{min}}{yw_{max} - yw_{min}}$$

$$= s_y\, y_w + \frac{yw_{max} \cdot yv_{min} - yv_{max}\, yw_{min}}{yw_{max} - yw_{min}}.$$

$$\boxed{y_v = s_y \cdot y_w + t_y}$$

where $t_y = \dfrac{yw_{max} \cdot yv_{min} - yv_{max} \cdot yw_{min}}{yw_{max} - yw_{min}}$.

We can get/obtain the transformation from world co-ordinates to viewport co-ordinates with sequence

1) Scale the clipping window to size of viewport using fixed point position of (xw_{min}, yw_{min})
2) Translate (xw_{min}, yw_{min}) to (xv_{min}, yv_{min})

The scaling transformation in ① can be represented as $$S = \begin{bmatrix} s_x & 0 & xw_{min}(1-s_x) \\ 0 & s_y & yw_{min}(1-s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

The 2D matrix representation for translation of lower left corner of clipping window to lower-left viewport corner is

$$T = \begin{bmatrix} 1 & 0 & xv_{min} - xw_{min} \\ 0 & 1 & yv_{min} - yw_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

And the composite matrix representation for transformation to normalized viewport is

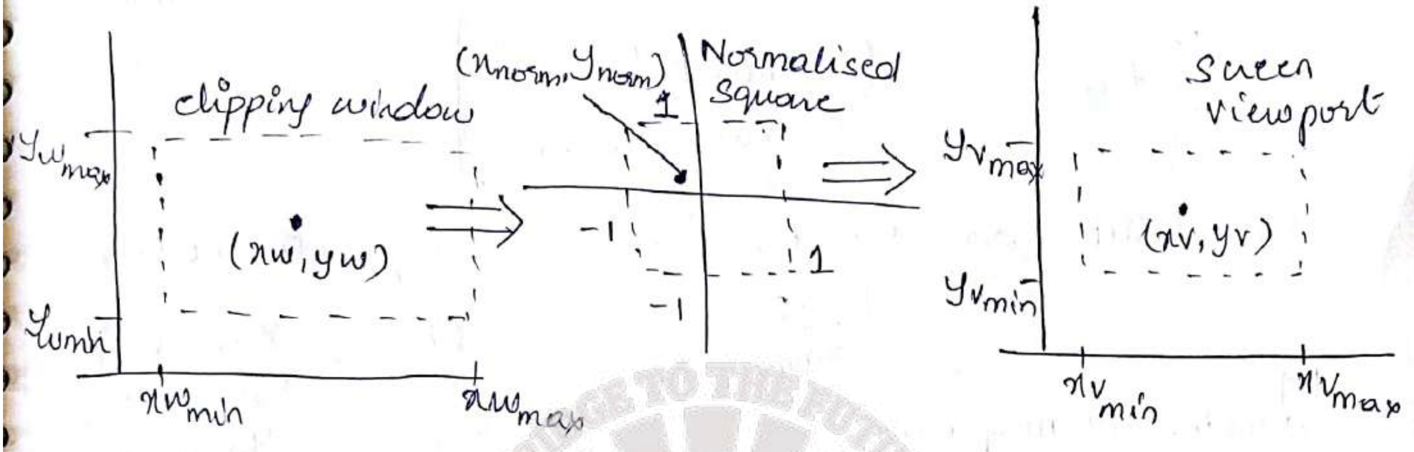
$$M_{window,\, normviewp} = T \cdot S = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Mapping Clipping Window into a Normalized Square

Transform the clipping window into a normalised square. → clip in normalized co-ordinates → then transfer the scene description to a viewport specified in screen co-ordinates.

- * Normalized co-ordinates are in range from -1 to 1.
- * clipping algorithms are standardized such that objects outside the boundaries $x = \pm 1$ & $y = \pm 1$ are detected & removed from scene description.

finally the viewing transformation, has the objects in the viewport positioned within display window.



A point (x_w, y_w) in the clipping window is mapped to normalized coordinate position (x_{norm}, y_{norm}), then to a screen co-ordinate position $(x_v\, y_v)$ in a viewport.

Consider the composite matrix.

$$M = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

for s_x & s_y, t_x & t_y, substitute -1 for xv_{min} and yv_{min}, $+1$ for xv_{max} & yv_{max}

where $$S_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} = \frac{1-(-1)}{xw_{max} - xw_{min}} = \frac{2}{xw_{max} - xw_{min}}$$

$$S_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} = \frac{1-(-1)}{yw_{max} - yw_{min}} = \frac{2}{yw_{max} - yw_{min}}$$

we also know.

$$t_x = \frac{xw_{max}\, xv_{min} - xv_{max} \cdot xw_{min}}{xw_{max} - xw_{min}} = \frac{xw_{max}(-1) - 1 \cdot xw_{min}}{xw_{max} - xw_{min}}$$

$$\boxed{t_x = \frac{-(xw_{max} + xw_{min})}{xw_{max} - xw_{min}}}$$

$$t_y = \frac{y_{wmax}\,y_{vmin} - y_{vmax}\,y_{wmin}}{y_{wmax} - y_{wmin}} = \frac{y_{wmax}(-1) - \overset{y_{wmin}}{\cancel{y_{vmax}}}(1)}{y_{wmax} - y_{wmin}}$$

$$\boxed{t_y = -\frac{(y_{wmax} + y_{wmin})}{y_{wmax} - y_{wmin}}}$$

Substitute S_x, S_y, t_x & t_y in composite matrix.

$$M_{window,normsquare} = \begin{bmatrix} \frac{2}{x_{wmax} - x_{wmin}} & 0 & -\frac{x_{wmax} + x_{wmin}}{x_{wmax} - x_{wmin}} \\ 0 & \frac{2}{y_{wmax} - y_{wmin}} & -\frac{y_{wmax} + y_{wmin}}{y_{wmax} - y_{wmin}} \\ 0 & 0 & 1 \end{bmatrix}$$

After clipping algorithms are applied, the normalized square with edge length equal to 2 is transformed into specified viewport by substituting −1 for x_{wmin} & y_{wmin} and +1 for x_{wmax} & y_{wmax}.

$$\therefore S_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}} = \frac{x_{vmax} - x_{vmin}}{1 - (-1)} = \frac{x_{vmax} - x_{vmin}}{2}$$

$$S_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}} = \frac{y_{vmax} - y_{vmin}}{1 - (-1)} = \frac{y_{vmax} - y_{vmin}}{2}$$

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

$$= \frac{1(x_{vmin}) - x_{vmax}(-1)}{1 - (-1)} = \frac{x_{vmin} + x_{vmax}}{2}$$

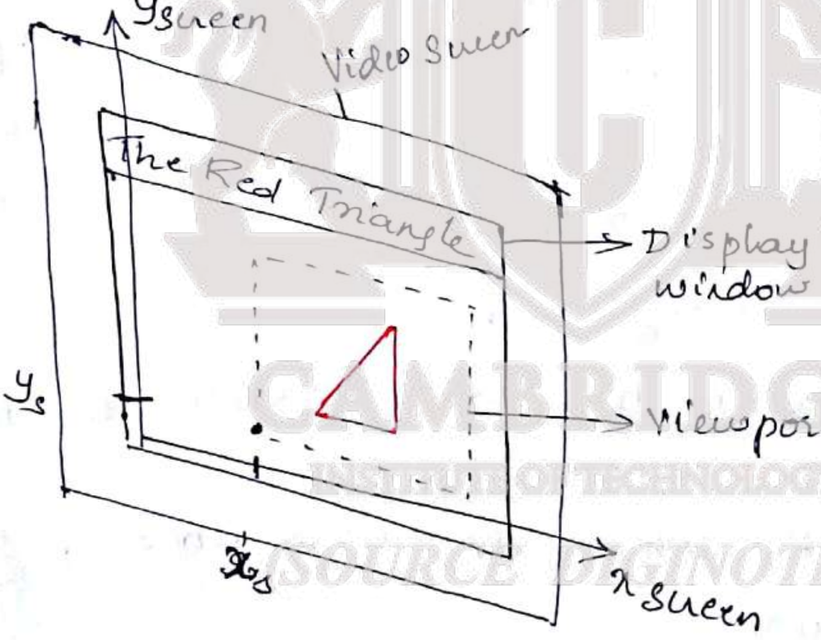
$$t_y = \frac{y_{wmax}\,y_{vmin} - y_{vmax}\,y_{wmin}}{y_{wmax} - y_{wmin}} = \frac{1(y_{vmin}) - y_{vmax}(-1)}{1 - (-1)}$$

$$t_y = \frac{y_{vmin} + y_{vmax}}{2}$$

Substitute s_x, s_y, t_x & t_y in composite matrix

$$M_{\text{normsquare, viewport}} = \begin{bmatrix} \frac{xv_{max} - xv_{min}}{2} & 0 & \frac{xv_{min} + xv_{max}}{2} \\ 0 & \frac{yv_{max} - yv_{min}}{2} & \frac{yv_{min} + yv_{max}}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

- The last step ~~of~~ in the viewing process is to position the viewport area in the display window
- Also ~~the~~ choosing the aspect ratio of the viewport are to be same as the clipping window. if not objects may be stretched or contracted in the x or y directions.



Viewport at co-ordinate position (x_s, y_s) within a display window.

Clipping Algorithms

Any procedure that eliminates those portions of a picture that are either inside or outside of a specified region of space is referred to as clipping Algorithm or simply clipping. [Rectangle shape] - xw_{min}, xw_{max} yw_{min}, yw_{max}.

Application of clipping is in viewing pipeline - to extract the designated portion of a scene for display.

Clipping algorithms are applied in 2D viewing procedures to identify those parts of a picture that are within the clipping window & everything outside it is eliminated.

Two Dimensional Point clipping.

a point $P = (x, y)$ in 2D would be saved for display if the following inequalities are satisfied.

$$xw_{min} \leq x \leq xw_{max}$$

$$yw_{min} \leq y \leq yw_{max}$$

If these 4 inequalities is not satisfied, the point is clipped

Applications: with scenes involving clouds, sea foam, smoke or explosions...

Two Dimensional Line clipping

A line-clipping algorithm processes each line in a scene through a series of tests & intersection calculation to determine whether the entire line or any part is to be saved.

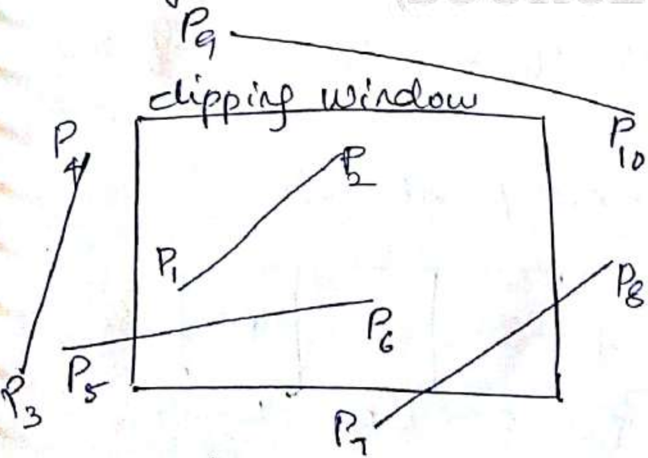
Expensive part of line clipping is in calculating the intersection position of line with the window edges. Hence major goal is to minimize the intersection calculation.

Inorder to do so, perform tests to determine whether a line segment is completely inside or completely outside the clipping window.

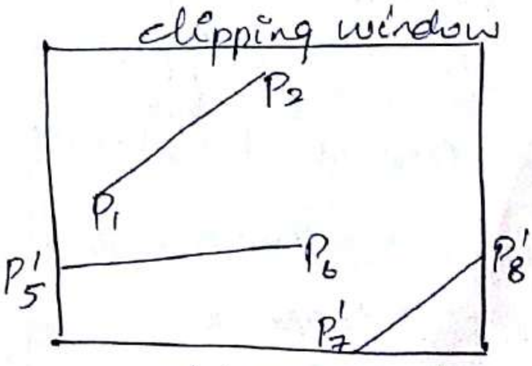
If we are unable to identify a line as complety inside or outside a clipping rectangle, we would then perform intersection calculations to determine whether any part of the line crosses the window Inter

By applying point clipping tests.

- when both the endpoints of a line segment are inside all 4 clipping boundary. Ex: P_1 to P_2, the line is complety inside the clipping window.
- when both endpoints of a line segment are outside any of the 4 clipping boundaries Ex: P_3 to P_4, that line is complety outside the window & is is eliminated from the scene.
- If both of these tests fail, the line segment intersects at least one clipping boundary & it may or may not cross into the interior of clipping window



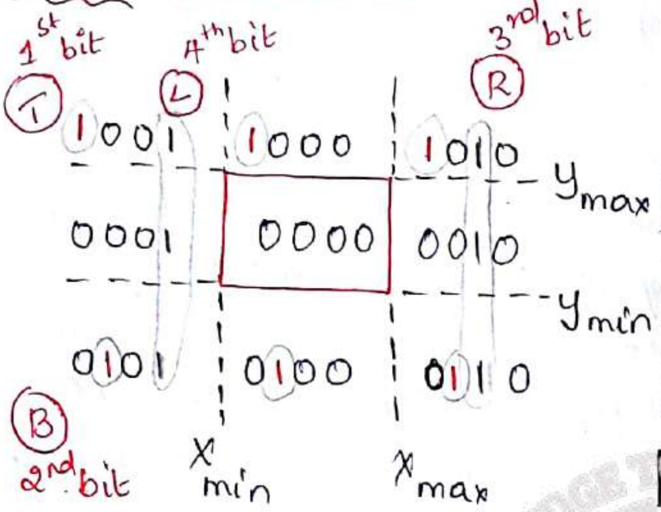
Befor Clipping



After clipping

Cohen-Sutherland Line Clipping

4- bit Code / T B R L
1^{st} 2^{nd} 3^{rd} 4^{th} bit



Bit Number	1	0
T 1^{st} bit	Above top edge $y > y_{max}$	below top edge $y < y_{max}$
B 2^{nd} bit	Below bottom edge $y < y_{min}$	Above bottom edge $y > y_{min}$
R 3^{rd} bit	Right of Right edge $x > x_{max}$	Left of right edge $x < x_{max}$
L 4^{th} bit	left of left edge $x < x_{min}$	Right of left edge $x > x_{min}$

- Every line endpoint in a picture is assigned a 4-digit binary value called region code.
- Each bit position is used to indicate whether the point is inside or outside one of the clipping-window boundaries.
- Each clipping window edge divides the 2D space into 9 regions.
- Any endpoint inside the clipping window has the region code "0000".

Step 1

① Calculate differences b/w endpoint co-ordinates & clipping boundaries

i.e Determine the bit values of 2 endpoints of the line to be clipped.

To determine & the bit value of any point use.

$b_1 = x - x_{min}$

$b_2 = x_{max} - x$

$b_3 = y - y_{min}$

$b_4 = y_{max} - y$

4	3	2	1
T	B	R	L
Top	bottom	Right	left

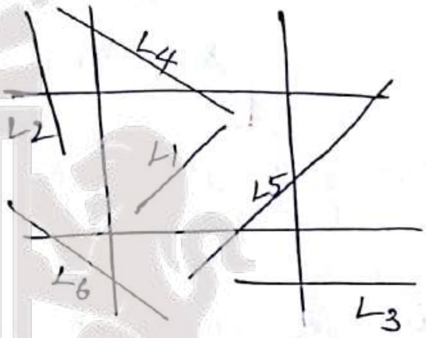
Step2:
Use these end points codes to locate the line.

Various possibilities

- If both endpoints codes are [0000], the line lies completely inside the box, no need to clip. This is the simplest case (eg. L_1).
- Any line has 1 in the same bit positions of both the endpoints, it is guaranteed to lie outside the box completely. (eg. L_2 & L_3)

(OR)

Perform AND operation for both the region codes if the result is not "0000" then line is completely outside. so line is rejected.



- Neither completely reject nor inside the box. lines L_4 & L_5 needs more processing.

Processing of lines neither completely inside or out for lines L_4, L_5, L_6

Basic idea is to clip parts of the line in any order. (consider from top to bottom).

Step3: Compute outcodes of both endpoints to check for trivial acceptance or rejection (AND Logic). if not so, obtain an endpoint that lies outside the box (at least one will?).

- Using the outcode, obtain the edge that is crossed by.

If the line crosses x_{min} / x_{max} then find

$$\boxed{y = y_0 + m(x - x_0)} \rightarrow \text{line eq}^n$$

Here $x = x_{min} / x_{max}$. the x value is set either to x_{min} or x_{max}.

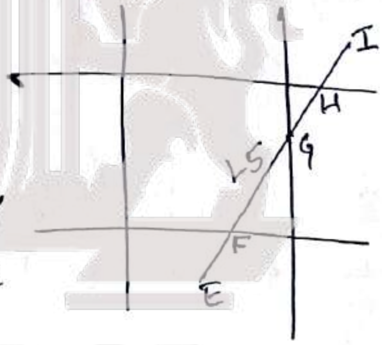
Slope of line $m = \dfrac{y_{end} - y_0}{x_{end} - x_0}$.

else if the line crosses y_{min} / y_{max}.

$$\boxed{x = x_0 + \frac{(y - y_0)}{m}}$$

Here y is set either to y_{min} or y_{max}.

obtain corresponding intersection points



* CLIP (replace the endpoint by the intersection point) . w.r.t the edge

* compute the outcode for the updated endpoint & repeat the iteration, till it is 0000.

eg: Take line L_5 (endpoint are E & I).

E has outcode 0100 .(to be clipped w.r.t bottom edge).

so EI is clipped to FI;

outcode of F is 0000;

But outcode of I is 1010;

clip (w.r.t topedge) to ge FH.

Outcode of H is 0010;

clip (with respect to right edge) to get FG.

Since outcode of G is 0000, display the final result FG.

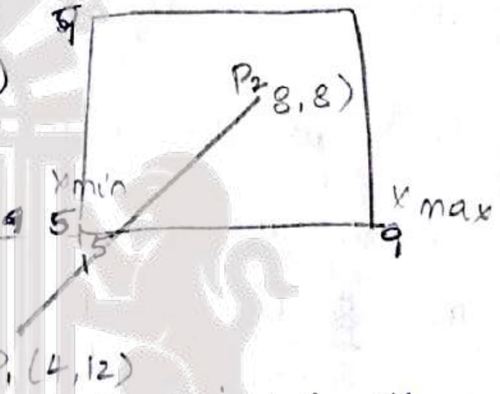
Example Cohen - Sutherland Algo.

Consider the window size from 5 to 9, clipp the following line (4,12), (8,8)

Soln : Line P_1 (4,12) to P_2 (8 8)

Region code for P_1 = 1001 (outside)

Region code for P_2 = 0000 (inside)



P_1 AND P_2

$$\begin{array}{r} 1001 \\ 0000 \\ \hline 0000 \end{array}$$

so the line is partially inside the window

As P_2 is 0000, this is inside the window, so no need to calculate new value.

P_1 is outside, so calculate the new value for P_1

$x_1 = 4$, $y_1 = 12$, $x_2 = 8$ $y_2 = 8$.

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{8-12}{8-4} = -1$$

Step 3 ① : Line is intersecting Xmin / X max, find y ?

$$y = y_0 + m(x - x_0)$$

$$y = 12 + -1(x-4)$$

Here line is intersecting at Xmin $x = x_{min} \Rightarrow x = 5$

$$y = 12 + (-1)(5-4) = 12 - 1$$

$y = 11$

So the new point is $(x, y) = (5, 11)$

Step 4 : check with condition.

$x_{min} \le x \le x_{max} \Rightarrow x_{min} \le 5 \le x_{max}$ (Yes)

$y_{min} \le y \le y_{max} \qquad y_{min} \le 11 \le y_{max}$ (NO).

Since x_{min} & $y_{min} = 5$

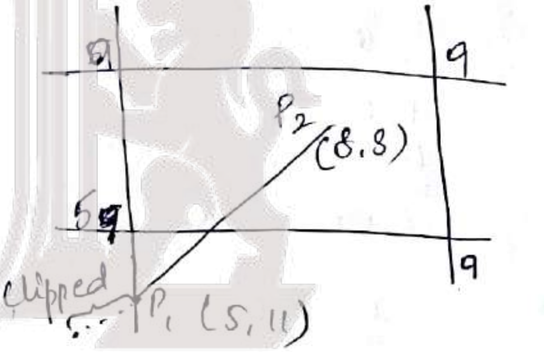
x_{max} & $y_{max} = 9$

The second condition is false, hence the ~~spect~~ step 3 is to be repeated again.

② So now the new line is from (5,11) to (8,8)

$x_1 = 5, y_1 = 11, x_2 = 8, y_2 = 8.$

$$m = \frac{8-11}{8-5} = -1$$



Now the following intersecting with y_{max}, find x.

$$x = x_1 + \frac{(y - y_1)}{m}.$$

Here $y = y_{max} = 9$.

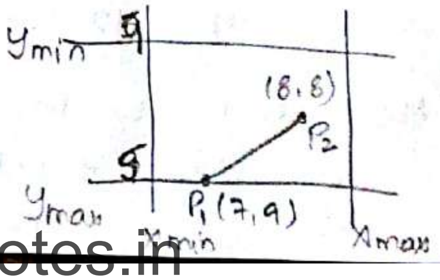
$$x = 5 + \frac{(9-11)}{m}$$

$$\underline{\underline{x = 7}}$$

Check for condition $x_{min} \le 7 \le x_{max}$

$y_{min} \le 9 \le y_{max}$.

Now the new line location (7, 9) to (8, 8)

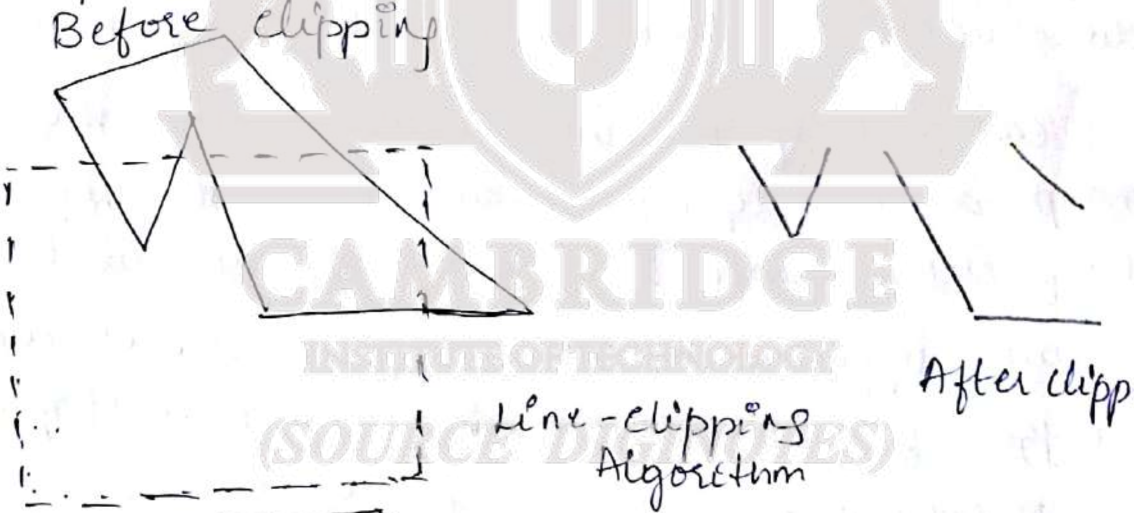


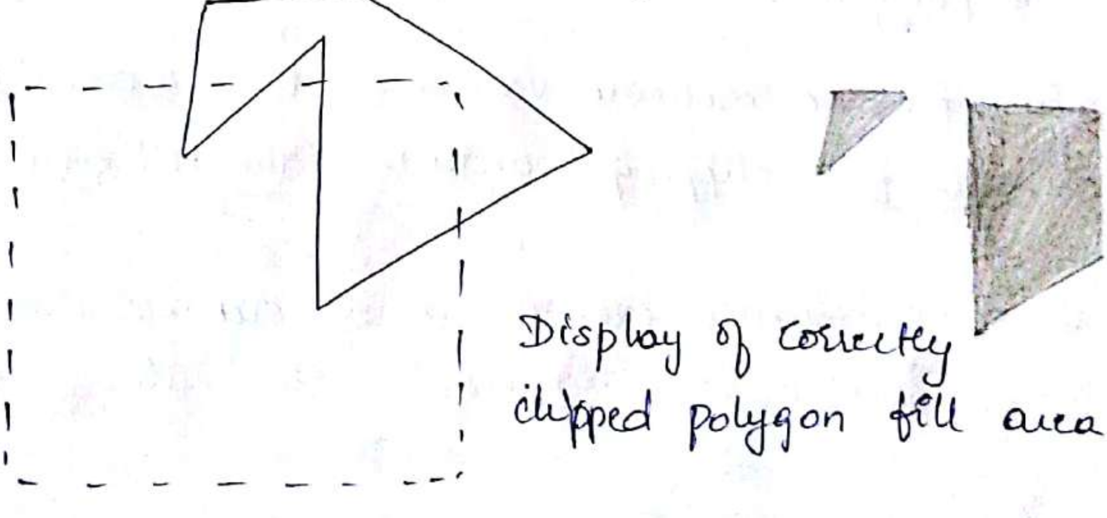
Polygon Fill-Area Clipping.

To clip a polygon fill area, it cannot directly apply a line-clipping method to the individual polygon edge (. will not produce a closed polyline)

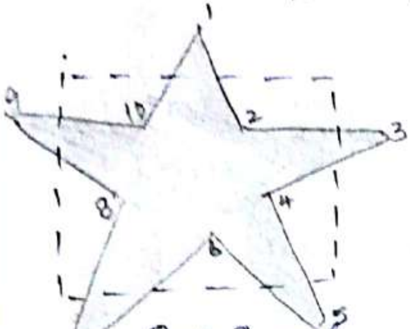
- A line clipper would often produce a disjoint set of lines with no complete information about how a closed boundary would form around the clipped fill area.

- A procedure is required that could output one or more closed polylines for the boundaries of the clipped fill area so that the polygon can be scan converted to fill the interiors with the assigned color or pattern.



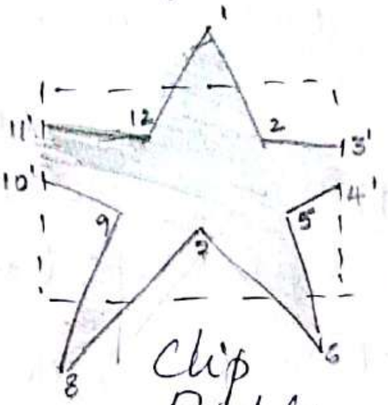

Display of correctly clipped polygon fill area.

Vertex number with "'" are the new list of vertices added

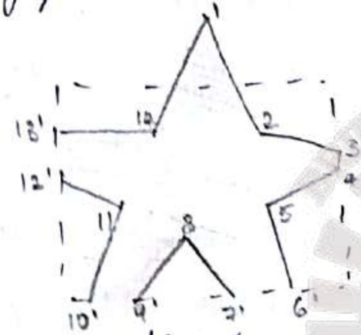


Original Polygon

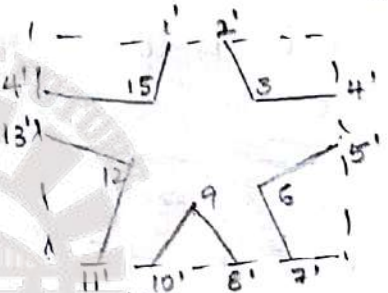
Clip left



Clip Right



Clip bottom



Clip Top.

Count of No. of vertices in the polygon after each clip.

* Polygon fill area is processed against borders of a clipping window using same general approach as in line clipping.

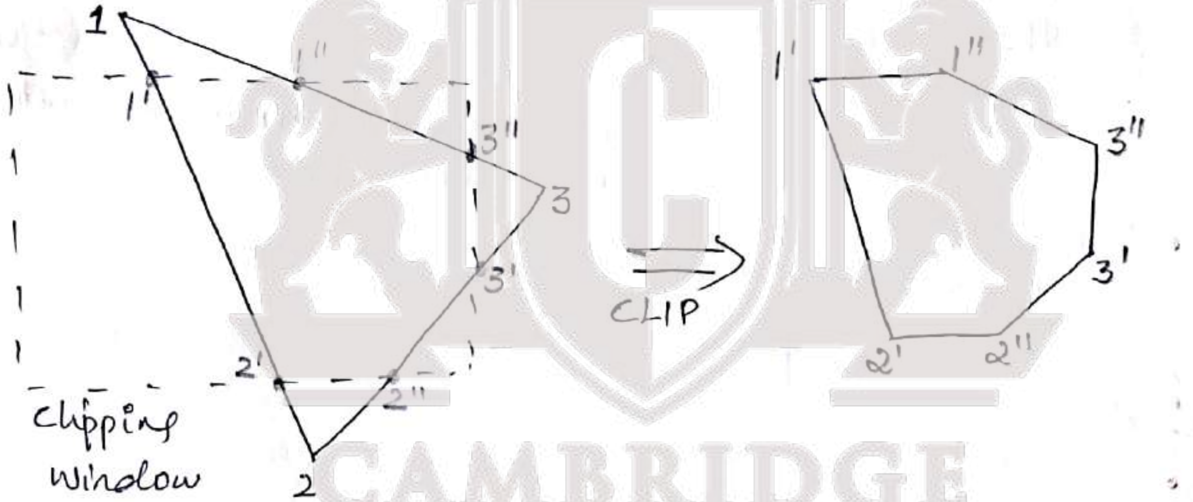
* The end points of the line segment are processed through a line-clipping procedure by constructing a new set of clipped endpoints at each clipping window boundary.

* As each clipping window edge is processed, we can clip a polygon fill area by determining the new shape for the polygon as shown in figure above.

* If min & max co-ordinate values for the fill area are inside all four clipping boundaries, the fill area is saved.

* If these co-ordinate extents are all outside any one of the clipping window borders, we eliminate the polygon.

- When the polygon fill area cannot be identified as being completely inside or completely outside the clipping window, then locate the polygon intersection positions with the clipping boundaries.
- One way to implement is to create a new vertex list at each clipping boundary, & then pass this new vertex list to the next boundary clipper
- The o/p of final clipping stage is the vertex list for the clipped polygon.

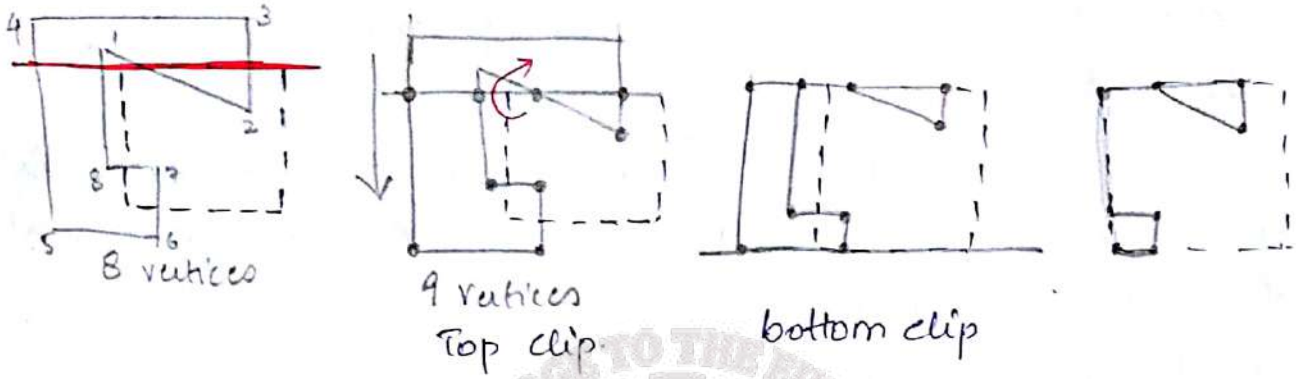


Sutherland - Hodgman Polygon Clipping

developed by Sutherland and Hodgman

- The method sends the polygon vertices through each clipping stage so that a single clipped vertex can be immediately passed to the next stage thus eliminating the need for an output set of vertices at each clipping stage.
- Final output is list of vertices that describes the edges of clipped polygon fill area.

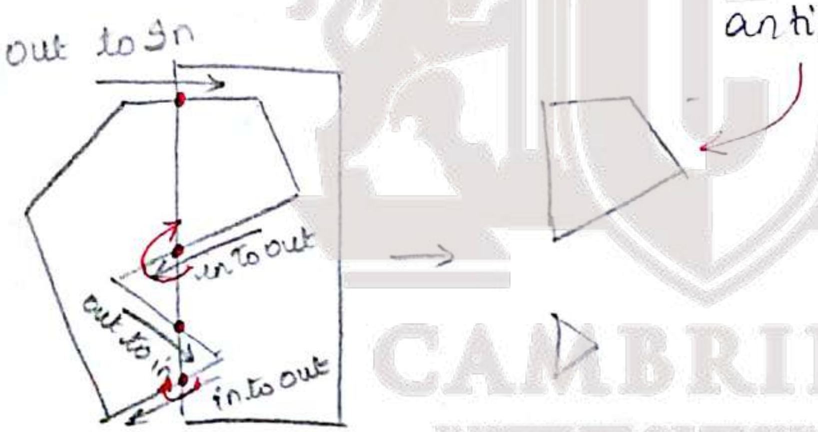
In case of multiple components - it cannot correctly generate 2 o/p polygons.



Now the o/p is
one pixel line joining

rather than

For this kind of problem
1) For out → in pair, follow the polygon boundary
2) For in → out, follow the window boundary in clock or anticlock wise.



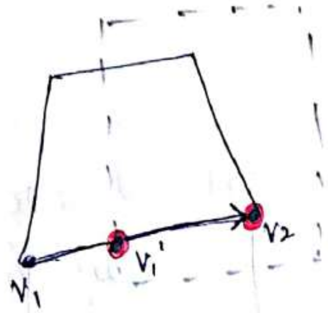
4 possible cases

case 1 : first edge endpoint is outside the clipping boundary & second endpoint is inside.

case 2 : Both endpoints are inside the clipping boundary.

case 3 : first endpoint is inside the clipping boundary & second endpoint is outside

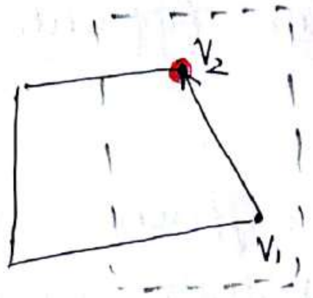
case 4 : Both endpoints are outside the clipping boundary.



out → in

case 1

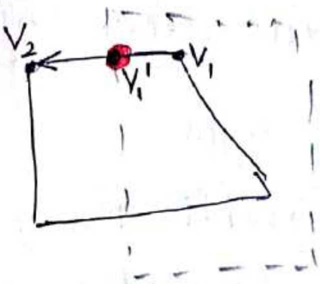
Output : V_1', V_2



in → in

case 2

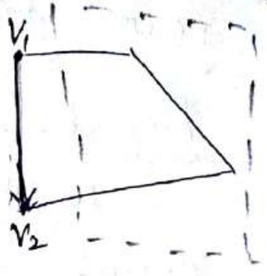
Output : V_2



in → out

case 3

output : V_1'



Out → out

case 4

Output : none

1. In case 1, both the intersection point of the polygon end with window border & the second vertex that is inside are sent to the next clipper.
2. In case 2, as both the vertices are inside the clipping window, only the second vertex (V_2 Position) is sent to the next clipper
3. In case 3, only the polygon edge-intersection position with the window border (marked in red) are sent to the next clipper
4. In case 4, both the vertices are outside, hence no vertices are sent to the next clipper.

Example to be reffered from textbook.

2.5 COLOR :

A visible color can be characterized by a function $C(\lambda)$ that occupies wavelengths from about 350 - 780nm, as shown in the fig. The value for a given wavelength λ in the visible spectrum gives the intensity of that wavelength in the color.

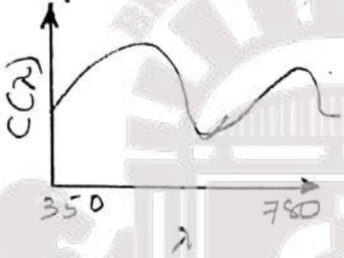
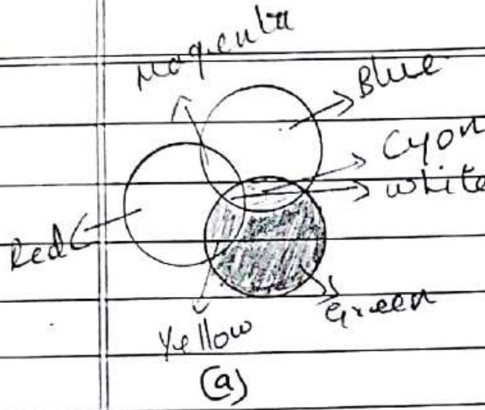


Fig: A color distribution

⇒ The human visual system has three types of cones responsible for color vision. Hence, our brains do not receive the entire distribution $C(\lambda)$ for a given color but rather three values - the **tristimulus values** - that are the responses of the three types of cones to the color. This reduction of a color to three values leads to the **basic tenet of three - color theory.**

*** If two colors produce the same tristimulus values, then they are visually indistinguishable ***



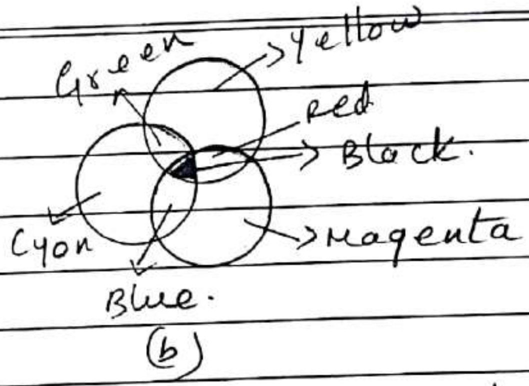


Fig: Color formation (a) Additive color (b) Subtractive color.

i) The CRT is one example of **additive color** where the primary colors add together to give the perceived color. Other examples of additive color include projectors and slide (+ve) film. In such systems the primaries are usually red, green, and blue.

With additive color, primaries add light to an initially black display yielding the desired color.

ii) For processes such as commercial printing and painting, a **subtractive color model** is more appropriate.

Here we start with a white surface such as a sheet of paper. colored pigments remove color components from light that is striking the surface.

In subtractive system, the primaries are usually the **complimentary colors**: cyan, magenta, and yellow **(CMY)** as shown in the fig.

2.5.1 RGB Color

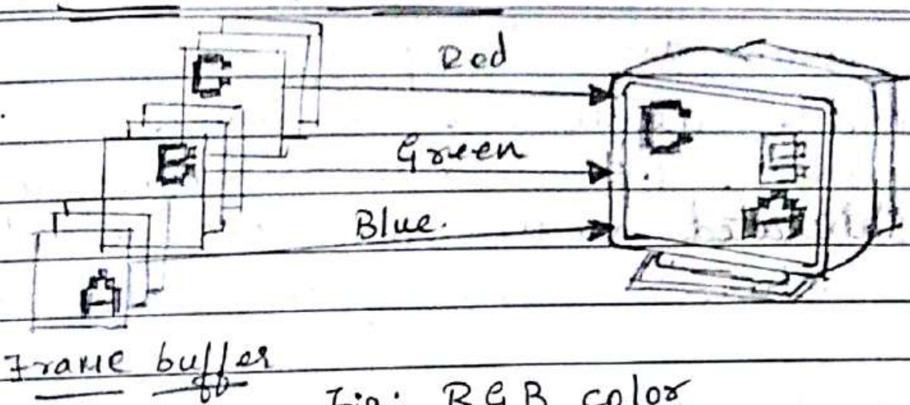


Fig: RGB color

2) In a three-primary color, additive-color RGB system, there are conceptually separate buffers for red, green, and blue images. ~~Ea~~ Each pixel has separate red, green, and blue components that correspond to locations in memory as shown in the above fig.

2) In a typical system, there might be a 1280 x 1024 array of pixels, and each pixel might consist of 24 bits (3 bytes): 1 byte for each of red, green, and blue.

2) With present commodity graphics cards having from 128MB to 512MB of memory, there is no longer a problem of storing and displaying the contents of the frame buffer at video rates.

=> A natural technique is to use the color cube and to specify color components as numbers between 0.0 and 1.0, where 1.0 denotes the maximum (or **saturated** value), and 0.0 denotes a zero value of that primary.

=> In OpenGL, we use the color cube as follows.

Ex:- To draw in red, we issue the following function call:

glColor3f(1.0, 0.0, 0.0);

The execution of this function will set the current drawing color to red.

=> Using 3f in glVertex function, conveys that we are using a three-color (RGB) model and that the values of the components are given as floats in C

=> Now, we shall be interested in a four-color (RGBA) system. The fourth color (A, or **alpha**) also is stored in the frame buffer as are the RGB values; it can be set with four-dimensional versions of the color functions.

*** uses for alpha, such as for creating fog effects or combining images. ***

=> **Alpha** value will be treated by OpenGL as either an opacity or transparency value → refer last page

=> Opacity values can range from fully transparent (A=0.0) to fully opaque. (A=1.0).

=> One of the first tasks that we must do in a program is to clear an area of the screen - a drawing window - in which to display our output.

We also must clear this window whenever we want to draw a new frame. By using the four-dimensional (RGBA) color system.

The function call

glClearColor (1.0, 1.0, 1.0, 1.0);

defines an RGB-color clearing color that is white, because the first three components are set to 1.0, and is opaque, because the alpha component is 1.0.

Hence the function glClear to make the window on the screen solid and white.

COLOR MODELS

(Refer [redacted])
color image process,
I & II] NPTEL

12.1 Properties of Light:

Light exhibits many different characteristics and we describe the properties of light in different ways in different contexts.

→ Electromagnetic spectrum:

In physical terms, color is electromagnetic radiation within a narrow frequency band.

Some of the other frequency groups in the electromagnetic spectrum are referred to as radio waves, microwaves, infrared waves, and X-rays.

Below figure shows the approximate frequency ranges for these various aspects of electromagnetic radiation.

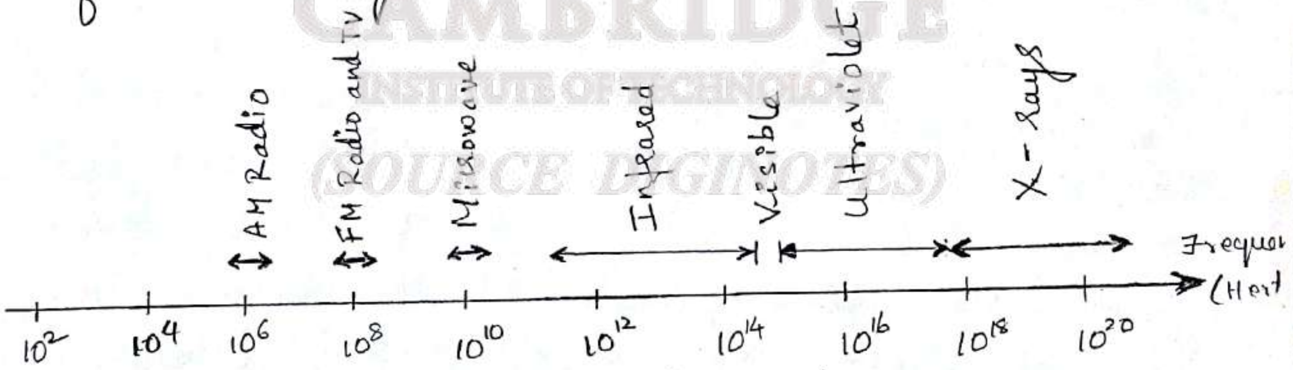


Fig: Electromagnetic spectrum

Note:

Each frequency value within the the visible region of the electromagnetic spectrum corresponds to a distinct spectral color.

At the low frequency end are the red colors, and at the high-frequency end are the violet colors. Actually, the human eye is sensitive to some frequencies into the infrared and ultraviolet bands.

Spectral color range from shades of red through orange and yellow, at the low-frequency end, to shades of green, blue, and violet at the high end.

A light source such as the sun or a standard household light bulb emits all frequencies within the visible range to produce white light.

When white light is incident upon an opaque object, some frequencies are reflected and some are absorbed.

The combination of frequencies present in the reflected light determines what we ~~perce~~ perceive as the color of the object. If low frequencies are predominent in the reflected light, the object is described as red.

In this case, we say that the perceived light has a **dominent frequency** or (dominent wavelength) at the red end of a spectrum. The dominant frequency is also called the **hue**, or simply the **color** of light

⇒ Physcological characteristics of color

Other properties besides frequency are needed to characterize our perception of light.

When we view a source of light, our eyes respond to the color (or dominant frequence) and two other basic sensations

-> One of these we call the **brightness**, which corresponds to the ~~light~~ total light energy and can be quantified as the luminance of the light. [Section 10.3]

-> The third perceived characteristics is called the purity, or the **saturation** of the light. Purity describes how close a light appears to be to a pure spectral color, such as red.

Pastels and Pale colors have low purity and they appear to be nearly white.

Another term, **chromaticity**, is used to refer collectively the two properties describing color characteristics: purity and dominant frequency (hue).

12.2 COLOR MODELS

Any method for explaining the properties or behavior of color within some particular context is called a color model.

Primary Colors:

When we combine the light from two or more sources with different dominant frequencies, we can vary the amount (intensity) of light from each source to generate a range of additional colors.

This represents one method for forming a color model.

The hues ~~to~~ (color) that we choose for the source are called the **primary colors**, and the color gamut for the model is the set of all colors that we can produce from the primary colors.

Two primaries that produce white are referred to as **complementary colors**.

Ex:- of complementary color pairs are red and cyan, green and magenta, and blue and yellow.

No finite set of real primary colors can be combined to produce all possible visible colors.

A ~~mixing~~ mixture of one or two of the primaries with the fourth color can be used to match some combination of the remaining primaries.

→ Intuitive Color Concepts:

~~The pi~~

An Artist creates a color painting by mixing color pigments with white and black pigments to form the various shades, tints, and tones in the scene.

Starting with the pigments for a "pure color" ("pure hue") the artist adds a black pigment to produce different **shades** of that color. the more black pigment, the darker the shade.

Similarly different **tints** of the color are obtained by adding a white pigment to the original color, making it lighter as more white is added.

Tones of the color are produced by adding both black and white pigments.

It is generally much easier to think of creating a pastel red color by adding white to pure red and producing a dark blue color by adding black to pure blue.

Therefore, graphics packages providing color palettes to a user often employ two or more color models.

12.4. RGB Color Model

According to the tristimulus theory of vision, our eyes perceive color through the stimulation of three visual pigments in the cones of the retina.

By comparing intensities in a light source, we perceive the color of the light. This theory of vision is the basis for displaying color output on a video monitor using the three primaries red, green, and blue, which is referred to as the **RGB color Model**

We can represent this model using the unit cube defined on R, G, B axes as shown in the fig

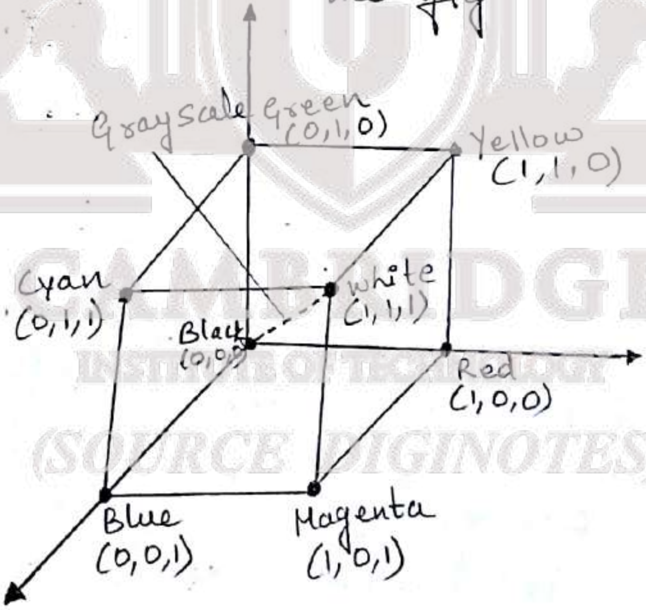


Fig: RGB color model

Any color within the unit cube can be described as an additive combination of the three primary colors.

The origin represents black and the diagonally opposite vertex, with coordinates (1,1,1) is white.

Vertices of the cube on the axes represent the primary colors, and the remaining vertices are the complementary color points for each of the primary colors.

The RGB color scheme is an additive model.

Each color point within the unit cube can be represented as a weighed vector sum of the primary colors, using unit vectors **R**, **G**, and **B**:

$$c(x) = (R, G, B) = R\mathbf{R} + G\mathbf{G} + B\mathbf{B}$$

where parameters R, G and B are assigned values in the range from 0 to 1.0.

For Ex:- ① The magenta vertex is obtained by adding maximum red and blue values to produce the triple (1, 0, 1) and ② white at (1, 1, 1) is the sum of the maximum values for red, green and blue.

③ Shades of gray are represented along the main diagonal of the cube from the origin (black) to the white vertex.

Points along ~~side~~ this diagonal have equal contributions from each primary color, and a gray shade halfway b/w black and white is represented as (0.5, 0.5, 0.5).

Example for RGB color model is TV, computer monitor, where the color displays on the black screen

12.6 The CMY AND CMYK Color Models

Hard copy devices, such as printers and plotters, produce a color picture by coating a paper with color pigments.

We see the color patterns on the paper by reflected light, which is a subtractive process.

The CMY Parameters :->

A subtractive color model can be formed with the three primary colors cyan, magenta and yellow.

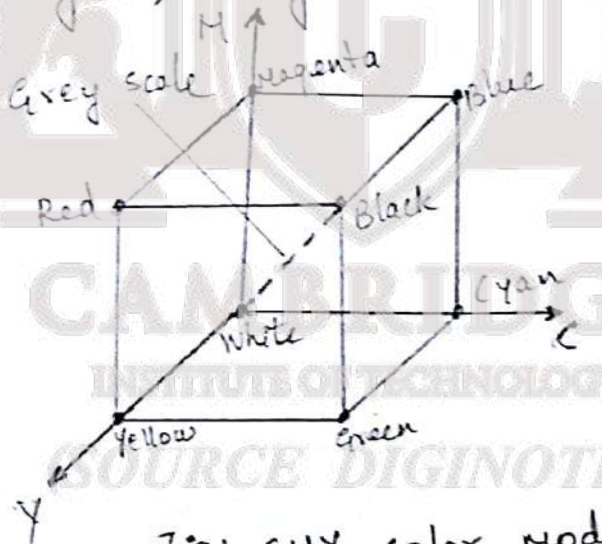


Fig: CMY color model

Positions within the unit cube are described by subtracting the specified amounts of the primary colors from white.

Cyan can be described as a combination of green and blue.

Th~~ese~~

Therefore when white light is reflected from cyan colored ink, the reflected light contains only the green and blue components, and the red component is absorbed or subtracted by the ink.

Similarly magenta ink subtracts the green component from incident light, and yellow subtracts the blue component.

A unit cube represents for the CMY model is illustrated in the above figure.

In the CMY model, the spatial position (1, 1, 1) represents black, because all components of the incident light are subtracted. The origin represents white light.

Equal amounts of each of the primary colors produce shades of gray along the main diagonal of the cube.

A combination of cyan and magenta ink produces blue light, because the red and green components of the incident light are absorbed. etc. Explain the cube

The CMY printing process often uses a collection of four ink dots, which are arranged in a close pattern somewhat as an RGB monitor uses three phosphor dots.

Thus, in practice, the CMY color model is referred to as the **CMYK model**, where K is the black color parameter.

One ink dot is used for each of the primary colors (cyan, magenta, and yellow), and one ink dot is black.

Illumination models

An illumination model, also called a **lighting model** (and sometimes referred to as a shading model), is used to calculate the color of an illuminated position on the surface of an object

lighting model
shading model
light source

10.1 LIGHT SOURCES

an object as light source & light reflector
properties → position, color, direction

=> Any object that is emitting radiant energy is a **light source** that contributes to the lighting effects for other object in a scene.

=> In some applications, however, we may want to create an object that is both a light source and a light reflector.

For Ex:- ~~A~~ plastic globe surrounding a light bulb both emits and reflects light from the surface of the globe.

=> A light source can be defined with a number of properties. We can specify its position, the color of the emitted light, the emission direction, and its shape.

If the source is also to be a light-reflecting surface, we need to give its reflectivity properties.

=> In addition, we could set up a light source that emits different colors in different directions.

For Ex:- we could define a light source that emits a red light on one side and a green light on the other side.

2) We assign light-emitting properties using a single value for each of the RGB color components, which we can describe as the amount, or the "intensity", of the color component.

i) Point Light Sources:

The simplest model for an object that is emitting radiant energy is a **point light source** with a single color, specified with three RGB components.

We define a point source for a scene by giving its position and the color of the emitted light, As shown in fig below.

Eg:- Mobile torch

Fig: Diverging ray paths from a point light source

In this the light rays are generated along radially diverging paths from the single color source position.

ii) Infinitely Distant Light Sources:

This light-source model is a reasonable approximation for sources whose dimensions are small compared to the size of objects in the scene.

ii) Infinitely Distant light Sources :

A large light source, such as the sun, that is very far from a scene can also be approximated as a point emitter, but there is little variation in its directional effects.

sun
little variation in direction
constant - light path from distant light source

The light path from a distant light source ~~by assigning~~ to any position in the scene is nearly constant, as illustrated in the below fig.

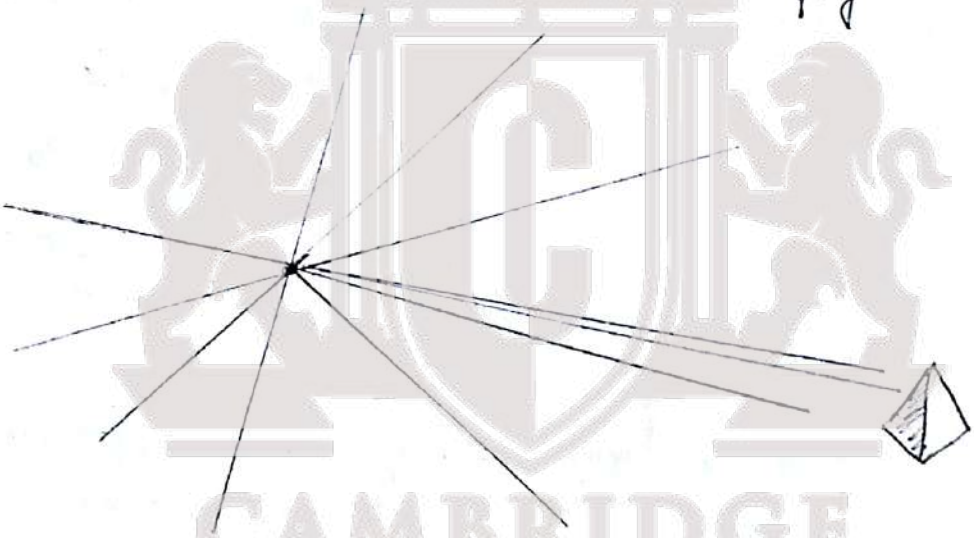


Fig: Light rays from an infinitely distant light source illuminate an object along nearly parallel light paths.

iii) Radial Intensity Attenuation:

As radiant energy from a light source travels outwards through space, its amplitude at any distance d_l from the source is attenuated by the factor $1/d_l^2$.

This means that a surface close to the light source ~~is attenuated by the factor~~ receives a higher incident light intensity from that source than a more distant surface.

Therefore, to produce realistic lighting effects, we should take this intensity attenuation into account. Otherwise all surfaces are illuminated with the same intensity from a light source, and undesirable display effects can result.

iii) Directional Light Sources and Spot light eff

A local light source can easily be modified to produce a directional, or spotlight beam of light.

One way to set up a directional light source is to assign it a vector direction and an angular limit θ_l measured from that vector direction, in addition to its position and color.

This defines a conical region of space with the light-source vector direction along the axis of the cone as shown in the fig below

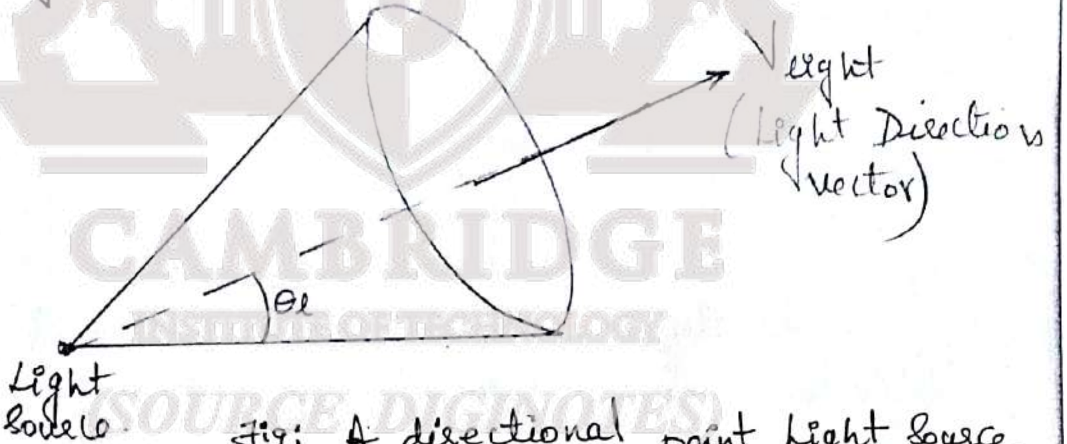


Fig: A directional point Light Source

→ The unit light-direction vector defines the axis of a light cone, and angle θ_l defines the angular extent of the circular cone

We can denote V_{light} as the unit vector in the light-source direction and V_{obj} as the unit vector in the direction from the light position to an object position. Then.

$$\boxed{V_{obj} \cdot V_{light} = \cos \alpha}$$

where

- Angle α is the angular distance of the object from the light direction vector.
- If we restrict the angular extent of any light cone so that $0° < \theta_t \leq 90°$, then the object is within the spotlight if $\cos\alpha > \cos\theta_l$ as shown in fig below
- But if $V_{obj} \cdot V_{light} < \cos\theta_l$, the object is outside the light cone

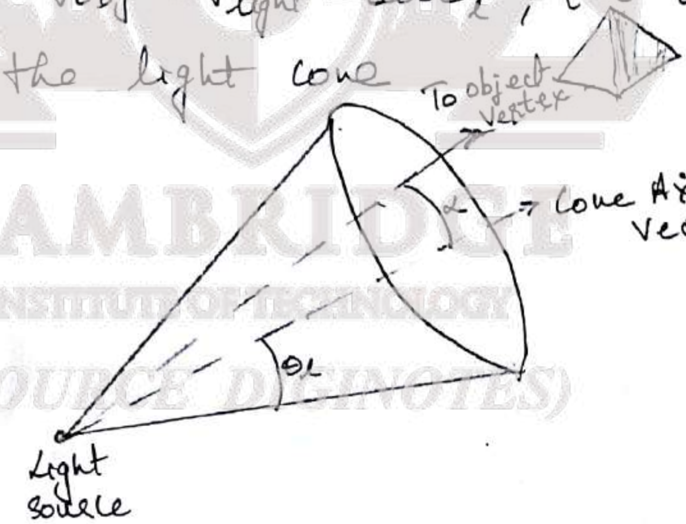


Fig: An object illuminated by a directional point light source.

iv) Angular Intensity Attenuation

For a directional light source, we can attenuate the light intensity angular about the source as well as radially out from the point-source position.

This allow us to simulate a cone of light that is most intense along the axis of the cone, with the intensity decreasing as we move farther from the cone axis.

10.3 BASIC ILLUMINATION MODELS

Accurate surface lighting models compute the results of interactions between incident radiant energy and the material composition of an object.

The empirical model described in this section produces reasonably good results, and its is implemented in most graphics systems.

=> Light-emitting objects in a basic illumination model are generally limited to point source.

However, many graphics packages provide additional functions for dealing with directional lighting (spot lights) and extended light sources.

Ambient Light :

In our basic illumination model, we can incorporate background lighting by setting a general brightness level for a scene.

This produces a uniform ambient lighting that is the same for all objects, and it approximates the global diffuse reflections from the various illuminated surfaces.

=> Assuming that we are describing monochromatic lighting effects, such as shades of grey, we designate the level for the ambient light in a scene with and intensity parameter I_a. Each surface in the scene is then illuminated ~~are simply a form of diffuse~~ with this background light.

=> Reflections produced by ambient-light illumination are simply a form of diffuse reflection, and they are independent of the viewing direction and the spatial orientation of a surface.

Diffuse Reflection :

We can model diffuse reflections from a surface by assuming that the incident light is scattered with equal intensity in all directions, independent of the viewing positions. Such surfaces are called ideal diffuse reflectors.

They are also ~~reffer~~ referred to as Lambertian reflectors, because the reflected radiant light energy from any point on the surface is calculated with Lambert's cosine law.

For Lambertion reflection, the intensity of light is the same over all viewing directions.

Assuming that every surface is to be treated as an ideal diffuse reflector (Lambertian), we can set a parameter Kd for each surface that determines the fraction of the incident light that is to be scattered as diffuse reflections. This parameter is called the diffuse-reflection coefficient or the diffuse reflectivity.

→ The diffuse reflection in any direction is then a constant, which is equal to the incident light intensity multiplied by the diffuse-reflection co-effecient.

=> For the background lighting effects, we can assume that every surface is fully illuminated by the ambient light Ia that we assigned to the scene.

Therefore, the ambient contribution to the diffuse reflection at any point on a surface is simply.

k-material

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

kd - diffuse reflection co-efficient, Ia - ambient light intensity.

Labertian Surface (ex: dull, matte surfaces, snow. chalk, projection & movie screens surfaces, uniformly painted walls.

Lambert's cosine law: The law states that the amount of radiant energy coming from any small surface area dA in a direction ϕ_N relative to the surface normal is proportional to $\cos\phi_N$.

$$\text{Intensity} = \frac{\text{radiant energy per unit time}}{\text{projected area}}$$

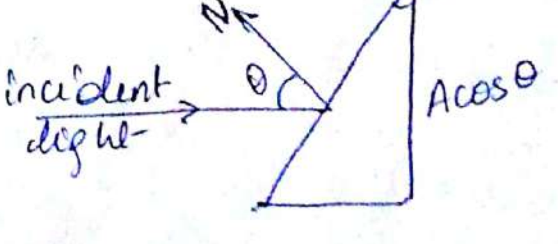
$$\propto \frac{\cos\phi_N}{dA \cos\phi_N}$$

$$= \text{constant}$$

angle of incidence - between the incoming light direction & surface normal as θ.

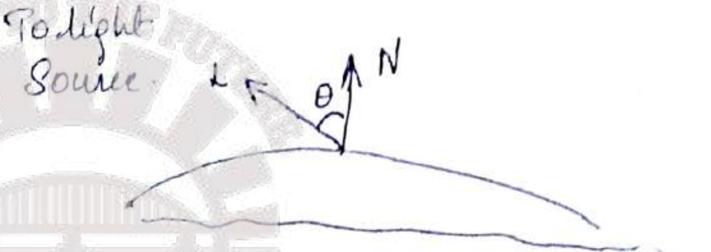
$$I_{l,diff} = K_d \cdot I_{l,incident}$$

$$= I_l \cos\theta \cdot K_d$$

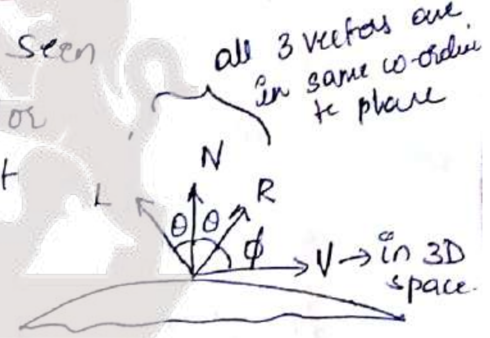


combining ambient & point-source intensity calculations to obtain an expression for total diffuse reflection at a surface position

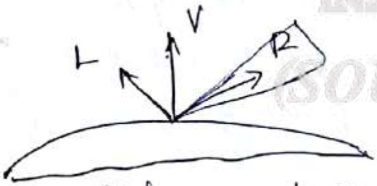
$$I_{diff} = \begin{cases} k_a I_a + k_d I_l (N.L) & \text{if } N.L > 0 \\ k_a I_a & \text{if } N.L \leq 0 \end{cases}$$



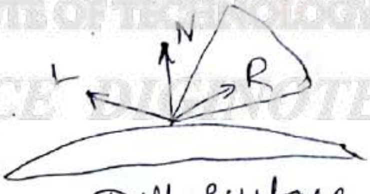
Specular Reflection and Phong Model



The bright spot or specular reflection seen on shiny surface is a result of total or near total reflection of incident light in concentrated region around the specular-reflection angle (equal to angle of Incident light)



Shiny surface (large ns)



Dull surface (small ns)

ns → specular reflection exponent

L, V → ½(h)

Intensity of specular reflection depends on material properties of surface & angle of Incidence.

W(θ) → specular reflection co-efficient.

I_l → intensity of light source, ϕ is viewing angle relative to specular-reflection direction R

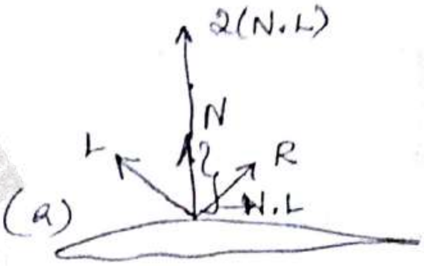
$$I_{spec} = W(\theta) . I_l \cos^{ns} \phi$$

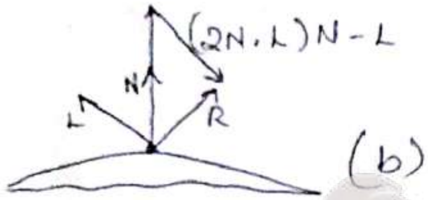
The direction for R, the reflection vector, can be computed from the directions for vectors L & N

$$R + L = 2(N \cdot L)N$$

and Specular-reflection vector is obtained as

$$R = (2N \cdot L)N - L$$

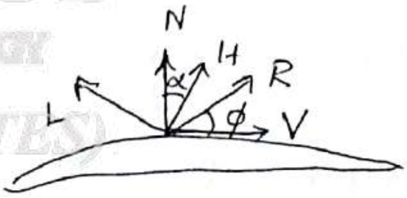




A simplified Phong model is obtained using halfway vector H between L and V to calculate the range of Specular reflection.

if we replace V.R in phong model with dot product N.H, the halfway vector is obtained

$$H = \frac{L+V}{|L+V|}$$



OpenGL Matrix Stacks

- ModelView, projection, texture & color (4 modes) can be selected with glMatrixMode function.

OpenGL maintains a matrix stack

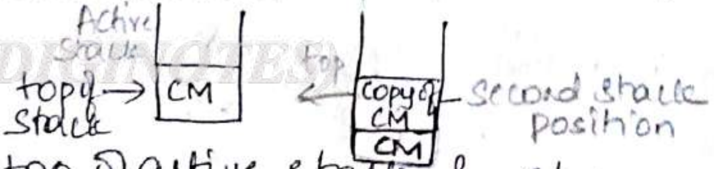
- initially each stack maintains/contains only identity matrix
- top of the stack is called "current matrix"
- As we perform viewing & geometric transformations, the top of modelview matrix stack is 4×4 composite matrix (combining viewing & various geometric transformation)
- modelview stack depth of 32 to save composite matrics for each created ~~with~~ by multiple views

glGetIntegerv (GL_MAX_MODELVIEW_STACK_DEPTH, stackSize);

other 3 stacks depth are of 2

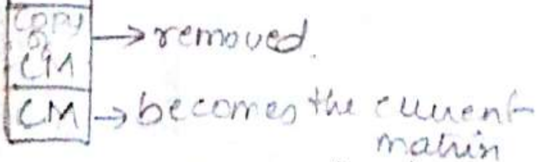
To find how many matrices are in stack currently

glGetIntegerv (GL_MODELVIEW_STACK_DEPTH, numMats);



* glPushMatrix();

copy the current matrix at top of active stack & store that copy in second stack position



* glPopMatrix();

destroys the matrix at top of stack & second matrix in the stack becomes the current matrix.

Prof. Supriya. S

Projections.

3D to 2D projection:

- Parallel Projections
  - Orthographic
  - Oblique.
- Perspective

Projection of a 3D object is defined by straight projection rays (projectors) emanating from the center of projection (COP) passing through each point of the object and intersecting the projection plane.

Perspective Projections.

Distant ~~of~~ from COP to projection plane is finite. The projections are not parallel. & we specify a center of projection (COP).

Center of projection is also called Perspective reference point.

Perspective foreshortening:

The size of the perspective projection of an object is inversely proportional ~~to~~ / varies inversely with the distances of object from the center of projection.

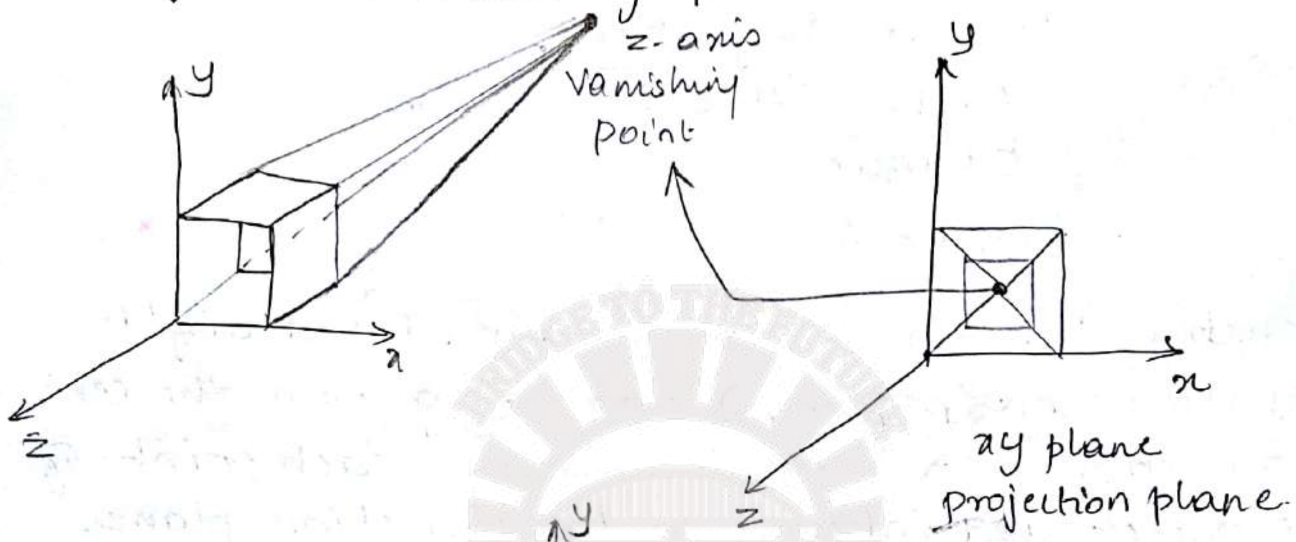
object far → looks smaller

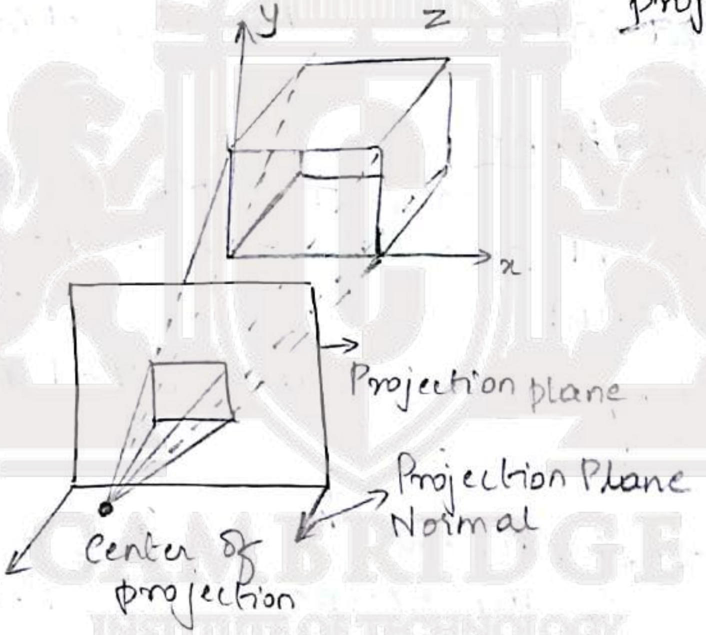
object close → looks larger.

Vanishing point: A railway track (||el lines), when you see a railway track from a fixed point on the track, as we observe the track to larger distances they observe to meet (converge) at a point even though we know the track lines are ||el. Such converging point is known as Vanishing point.

Prof. Supriya S

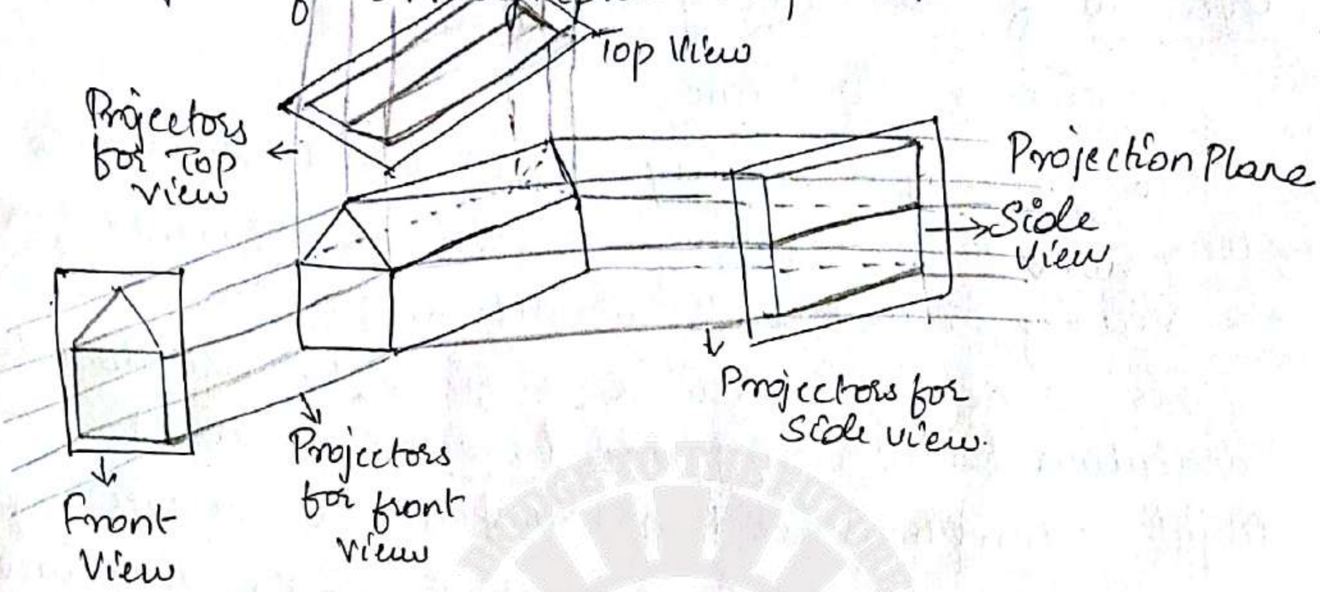
Def^n : The perspective projections of any set of ||el lines that are not parallel to the projection plane converge to a vanishing point.





Prespective projection

Example of Orthographic Projection



Axonometric orthographic projection.

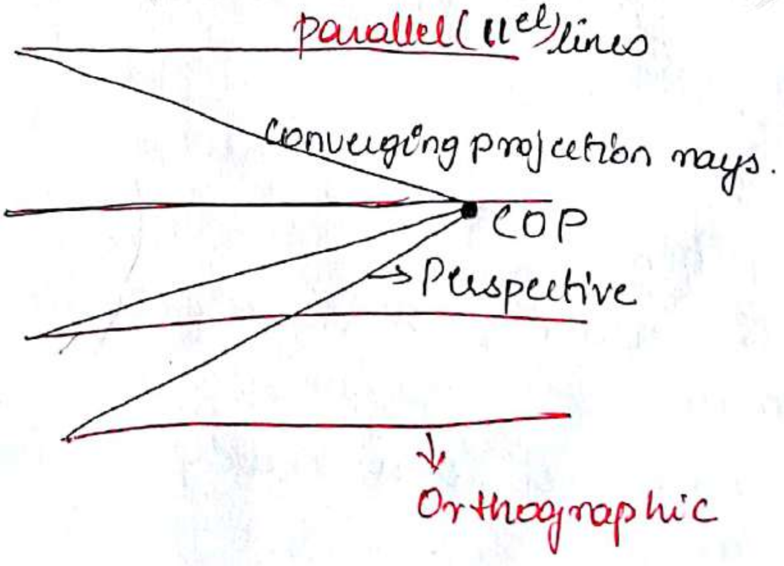
Use planes of projection that are not normal to a principal axis. (show multiple face of an object).

Isometric: Projection plane normal makes equal angles with each principle axis. DOP vector: [1 1 1]

Difference between Orthographic & Perspective.

Orthographic is infinite ($\perp^{lar}$ angle to projection plane)

Perspective is finite, COP is at a fixed point. (different angles to PP)



Prof. Supriya S

Overview of 3D viewing concepts

Viewing a 3D scene.

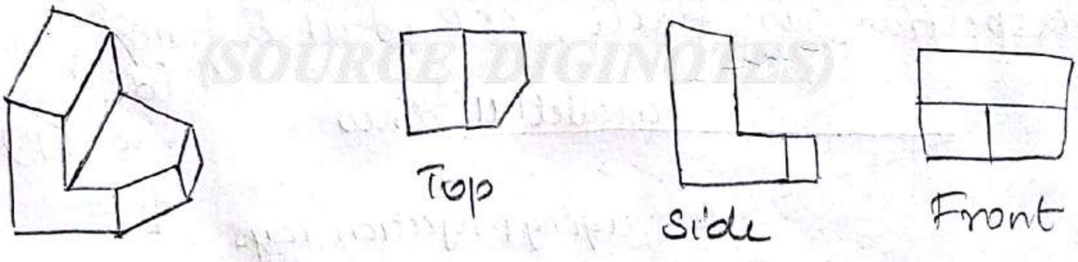
To obtain a display of 3D world-coordinate scene, we first set up a co-ordinate reference for the viewing or "camera" parameters.

This co-ordinate reference defines the position & orientation for a view plane (projection plane). Object descriptions are then transfered to the viewing reference co-ordinates & projected on to the view plane.

- View of an object on the output device is like a wireframe (outline), later lighting, surface rendering are applied to obtain realistic shading of visible surface.

Projections.

Parallel projection - project the points on the object surface along parallel lines onto a view plane.



Perspective projection -

causes objects farther from viewing position to be displayed smaller.

casuses objects nearer from viewing position to be displayed larger

Prof. Supriya S

Depth cueing

- depth information is important in 3D scene - for a viewing direction, which is front & which is back of each displayed object.

- A simple method to indicate depth in wire frame ~~mode~~ display is to vary the brightness of line segments based on their distances from the viewing position. Line closest to the viewing position are displayed with highest intensity, lines father away are displayed with decreasing intensity.

- Another method ~~of~~ of depth cueing is modelling the effect of the atmosphere on the percieved intensity (color) & of objects. distant objects - appear dimmer than near objects due to light scattering by dust particles, haze & smoke.

Identifying Visible Lines and Surfaces.

can clarify depth information/relationships using

① highlight the visible lines or to display them in a different color.

② display non-visible lines as dashed lines. or remove the non-visible lines from display. (removes info about shape of back surfaces).

when realistic scene is to be produced → back parts of objects are completely eliminated so that only visible surfaces are displayed.

Prof. Supriya S

Surface Rendering

Rendering the object surfaces using ~~ob~~ lighting conditions in scene & assigned surface characteristics.

Surface → transparent or opaque.
↳ smooth or rough

Surface rendering is combined with perspective & visible surface identification to generate a degree of realism.

lighting conditions → by specifying color, location of light source, & background illumination effects.

Exploded & Cutaway Views

→ used to show the internal structures & relationships of object parts.

Refer figures from textbook.

Stereoscopic Viewing

3D views can be obtained by reflecting a raster image from a vibrating, flexible mirror.

Stereoscopic devices present 2 views of scene, one for the left eye & other for right eye. The viewing positions correspond to the eye position of the viewer. It is displayed on alternate refresh cycles of raster Monitor.

Prof. Supriya S

The Three-Dimensional Viewing Pipeline

- Viewing position corresponds to where we would place a camera.
- choose viewing position according to what we want to display (front, back , side, top, bottom view of the scene)
- Orientation of the camera

Some of the viewing operations for a 3D scene are same as in 2D viewing pipeline as described in the below figure.

* 2D clipping window is used to select a view that is to be mapped to the viewport.
* 2D viewport is used to position a projected view of 3D scene on output device.

In 3D viewing, a ~~3~~ clipping window is positioned on a selected view plane & scenes are clipped against an enclosing volume of space defined as set of clipping planes.
The viewing position, view plane, clipping window & clipping planes are all specified within the viewing-co-ordinate reference frame.

① Once the scene has been modeled in world co-ordinates, a viewing co-ordinate system is selected & the description of scene is converted to viewing co-ordinates which defines viewing parameters including the position & orientation of the projection plane (view plane) - a camera film plane.

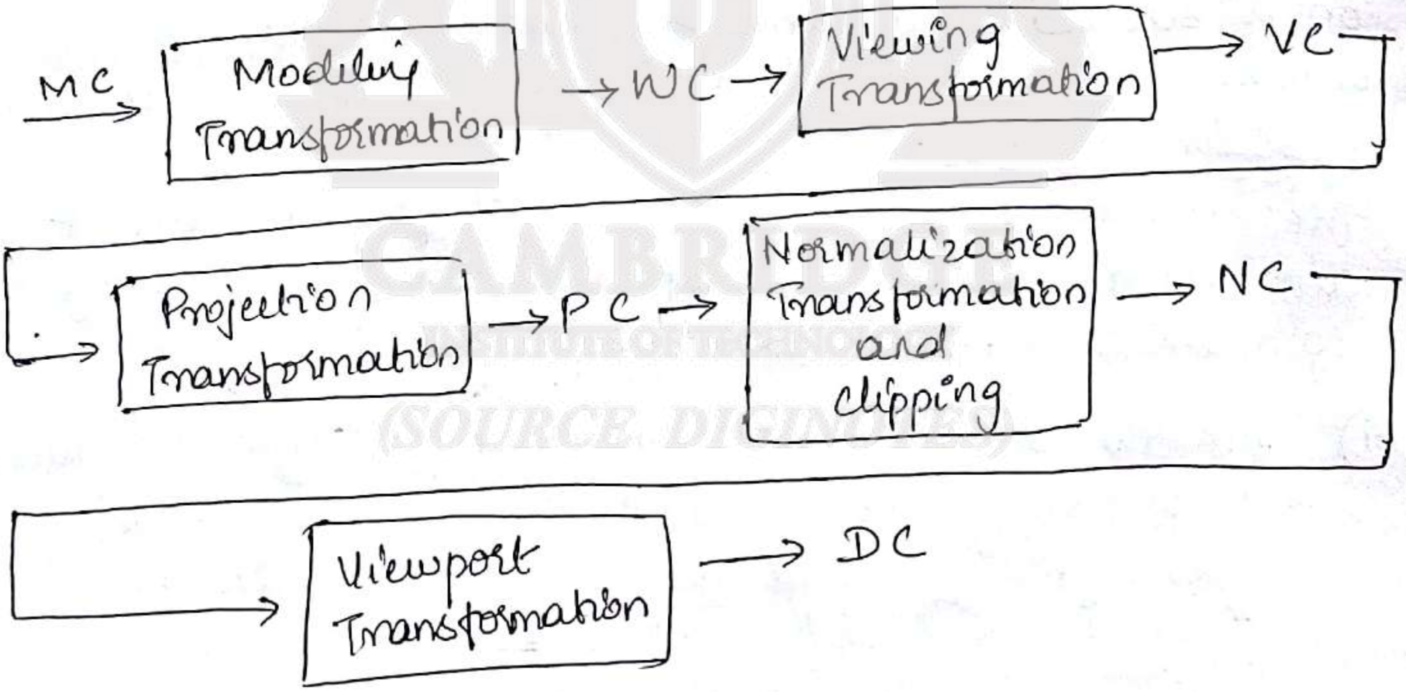
② A 2D clipping window, corresponding to a selected region is as camera lens, is defined on the projection plane & 3D clipping region is established. This clipping region is called <u>view volume</u> (its shape & size depends on dimensions of clipping window, type of projections ~~operation~~, direction of viewing)

③ Projection operations are performed to convert viewing co-ordinate description of scene to co-ordinate positions on projection plane.

④ Objects are mapped to normalized co-ordinates & object parts outside the view volume are clipped off.

⑤ finally to viewport transformations (other tasks such as identifying visible surfaces, apply surface-rendering)

⑥ Final step is to map viewport coordinates to device co-ordinates within a selected display window.



3D Transformation Pipeline

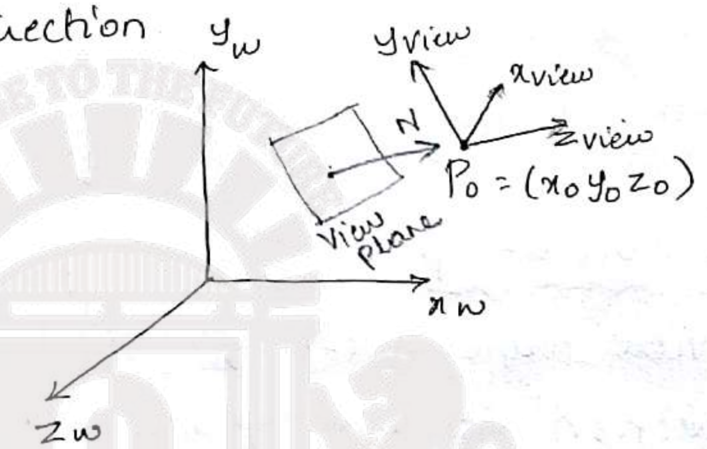
Prof. Supriya S

Three-Dimensional Viewing-coordinate Parameters

Select a world co-ordinate position $P_0 = (x_0 y_0 z_0)$ for a viewing origin, which is called the View point or viewing position (eye position or camera position)

A view-up vector 'V' which defines y_{view} direction.

z_{view} → viewing direction

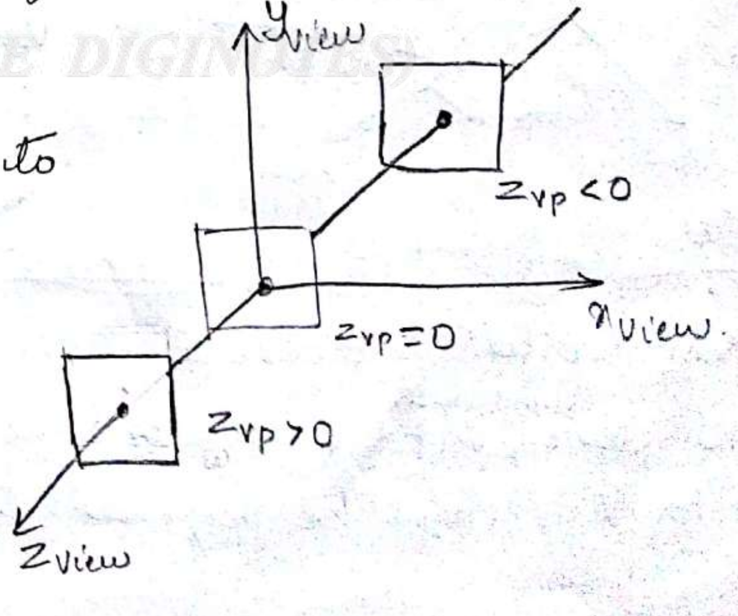


The View-Plane Normal Vector

* Viewing direction is along z_{view} axis, the view plane or projection plane is normally assumed to be perpendicular to z_{view} axis

* Orientation of view plane as well as direction of +ve z_{view} can be defined with a view-plane normal vector N

Viewplane is parallel to x_{view} & y_{view}



Vector N can be specified in various ways

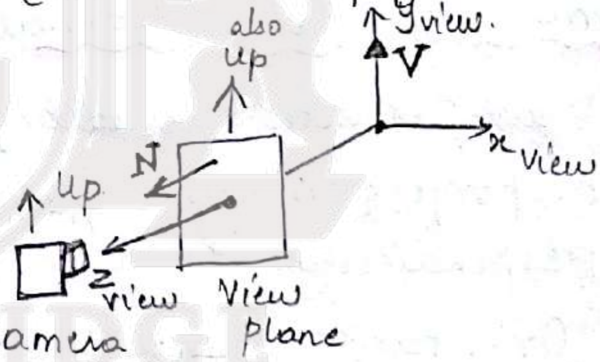
① the direction of N is defines as the line from world-coordinates to selected point position.

② or direction from a reference point P_{ref} to viewing origin P_0.

Reference point P_{ref} is refered to as look-at-point with viewing direction oppossite to direction of N. See figure 3 in below.

The View-up vector.

view plane vector Normal 'N' is choosen, the direction of view-up vector 'V' can also be set in the positive direction of y-axis (where the 'up' of the camera is)



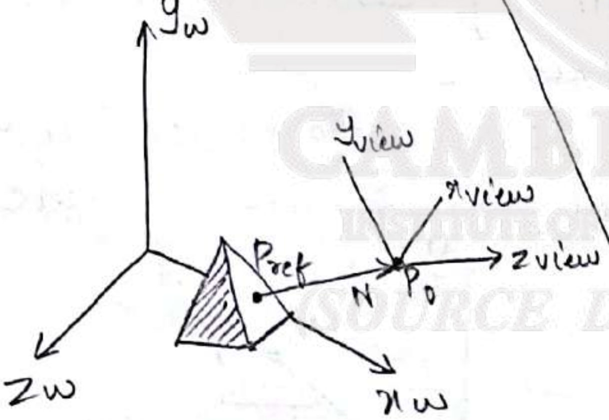


figure 3.

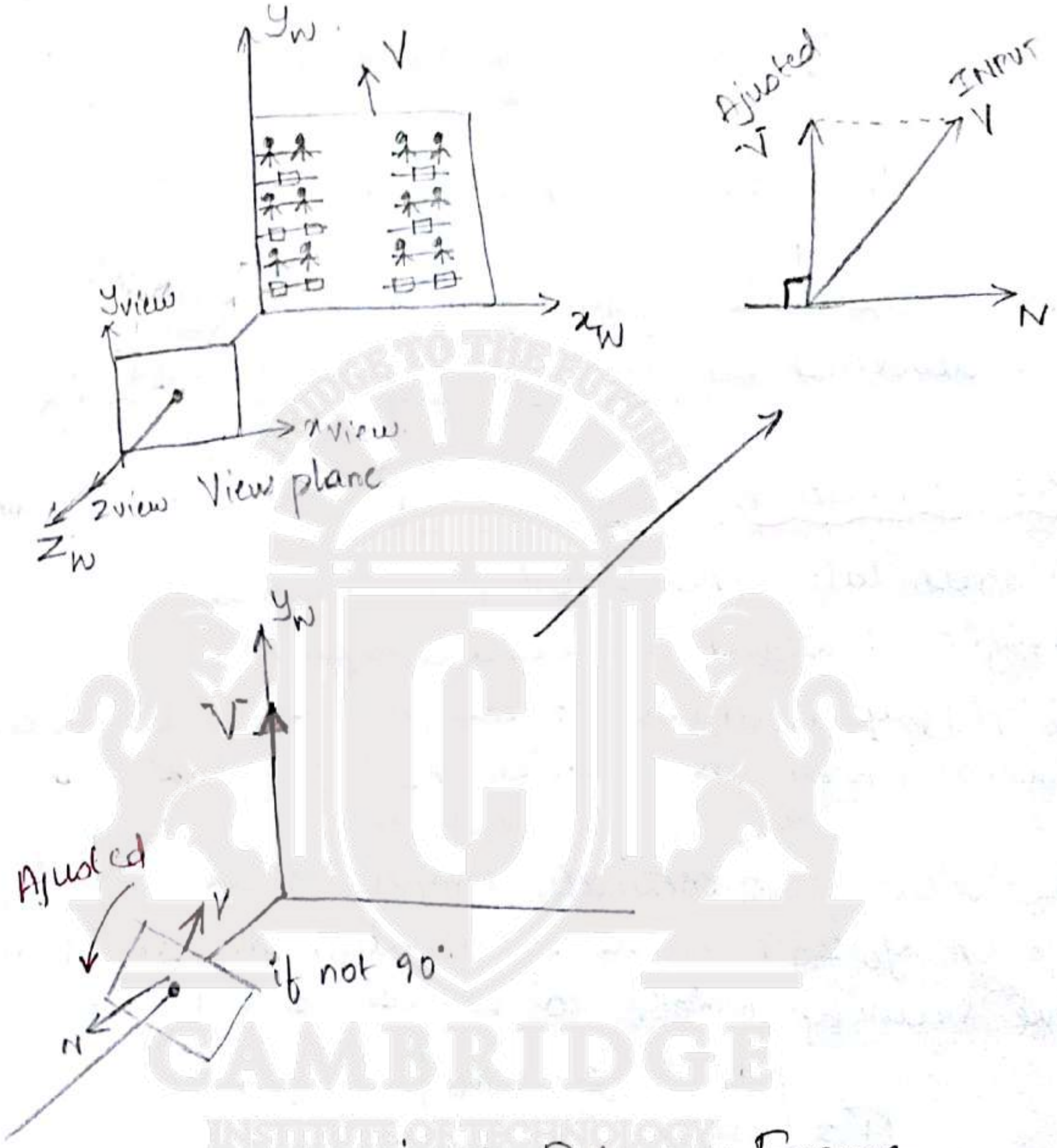
Usually V is defined by selecting position relative to world-coordinate origin. so that the direction for the view-up vector is from world co-ordinate origin to this selected position

View-plane normal vector V defines the direction of z_{view}-axis, vector V should be perpendicular to N.

Prof. Supriya S

Therefore viewing routines typically adjust the user-defined orientation of vector V.

Ex:



The uvn Viewing-coordinate Reference Frame.

View plane normal vector defines the direction for zview, & view up vector is used to obtain the direction for Yview, we need only to determine the direction for xview (U vector)

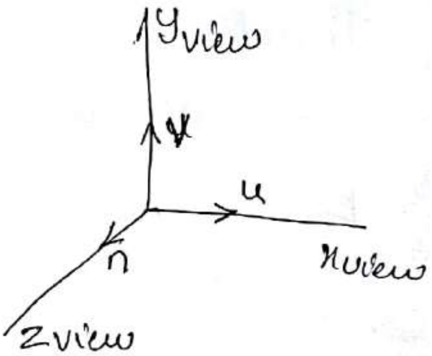
U vector is perpendicular to both N & V.

- taking the cross product of N and V.

$$n = \frac{N}{|N|} = (n_x, n_y, n_z)$$

$$u = \frac{V \times n}{|V|} = (u_x, u_y, u_z)$$

$$v = n \times u = (v_x, v_y, v_z)$$



The co-ordinate system formed with these unit vectors are described as uvn viewing-coordinate reference frame

Transformation from World to Viewing Coordinates

① Translate the viewing-coordinate origin to the origin of world co-ordinate system.

② Apply rotations to align the x_{view}, y_{view} & z_{view} axes with the world x_w, y_w & z_w axes respectively

The viewing co-ordinate origin is at world position $P = (x_0\ y_0\ z_0)$ ∴ The translation matrix translating the viewing origin to world origin is

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For rotation transformation, the unit vectors u, v & n are used to form composite rotation matrix.

$$R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{wc,vc} = R \cdot T$$

$$= \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} u_x & u_y & u_z & -x_0 u_x - y_0 u_y - z_0 u_z \\ v_x & v_y & v_z & -x_0 v_x - y_0 v_y - z_0 v_z \\ n_x & n_y & n_z & -x_0 n_x - n_y y_0 - z_0 n_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} u_x & u_y & u_z & -u \cdot P_0 \\ v_x & v_y & v_z & -v \cdot P_0 \\ n_x & n_y & n_z & -n \cdot P_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$-u \cdot P_0 = -x_0 u_x - y_0 u_y - z_0 u_z$

$-v \cdot P_0 = -x_0 v_x - y_0 v_y - z_0 v_z$

$-n \cdot P_0 = -x_0 n_x - y_0 y_n - z_0 n_z$

Prof. Supriya S

Projection Transformations.

① Orthogonal Projections

Distance from COP to projection plane is infinite – Parallel projection.

- * A transformation of object descriptions to a view plane along lines that are all parallel to the view plane normal vector N is called orthogonal projection
- * Projection lines are perpendicular to view plane.
- * commonly used to produce the front, side & top views of an object.
- * Front, side & rear orthogonal projections – called Elevators.
- * Top orthogonal projection is called plan view.

Refer figure from text book page no : 357 – 3rd edition

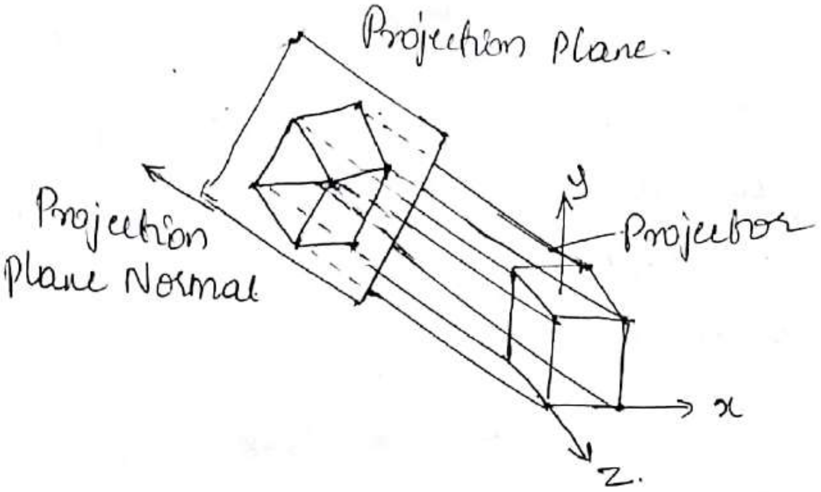
Anonometric & Isometric Orthogonal Projections

An Orthogonal projections formed that displays more than one face of an object. are called Anonometric orthogonal projections.

The most commonly used axonometric projection is the isometric projection.

Anonometric projections use planes of projection that are not normal to a principal axis (they therefore show multiple face of an object)

Isometric projection : projection plane normal makes equal angles with each principle axis



Example of ~~Isoso~~ Isometric projection

All 3 principal axes are foreshortened equally in an isometric projection, so that relative ~~projectio~~ proportions are maintained.

Clipping Window and Orthogonal - Projection View Volume

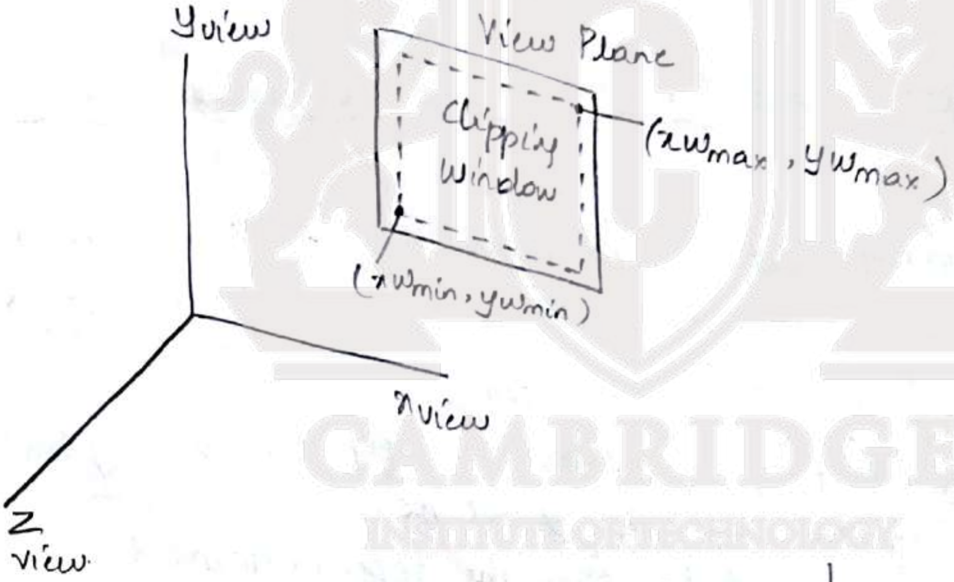
- For 3D viewing, the clipping window is positioned on the view plane with its edges parallel to xview & yview axis.
- The edges of clipping window specify the x & y limits - that display the part of the scene.
- These limits are used to form the top, bottom, & 2 sides of clipping region called orthogonal-projection view volume.
- Since the projection lines are perpendicular to the view, these four boundaries are planes that are also perpendicular to the viewplane.
- The extent of orthogonal view volume is limited in the zview direction by selecting positions for one or two additional boundary planes that are

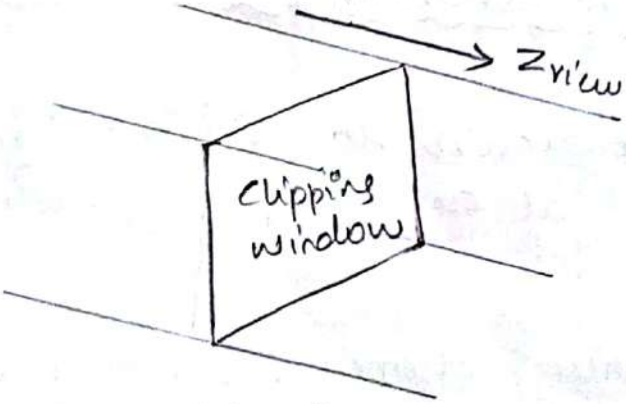
parallel to view plane. These two planes are called near-far clipping planes or front-back clipping planes.

- The near and far planes allow us to exclude objects that are in front of or behind the part of scene that we want to display.

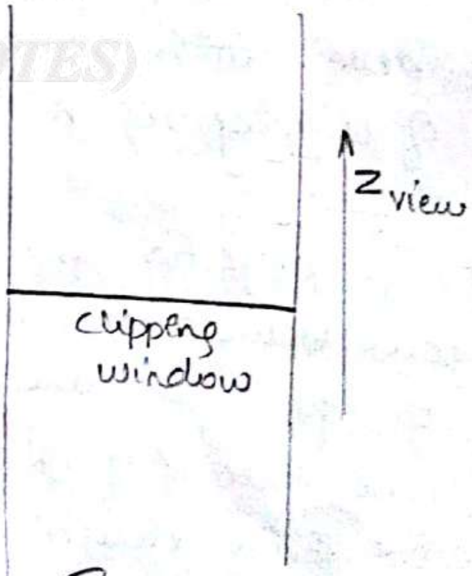
- $z_{far} < z_{near}$ so that the far plane is farther out along the negative z_{view} axis.

- when a near and far planes are specified, we obtain a finite orthogonal view volume which is rectangular parallelepiped.

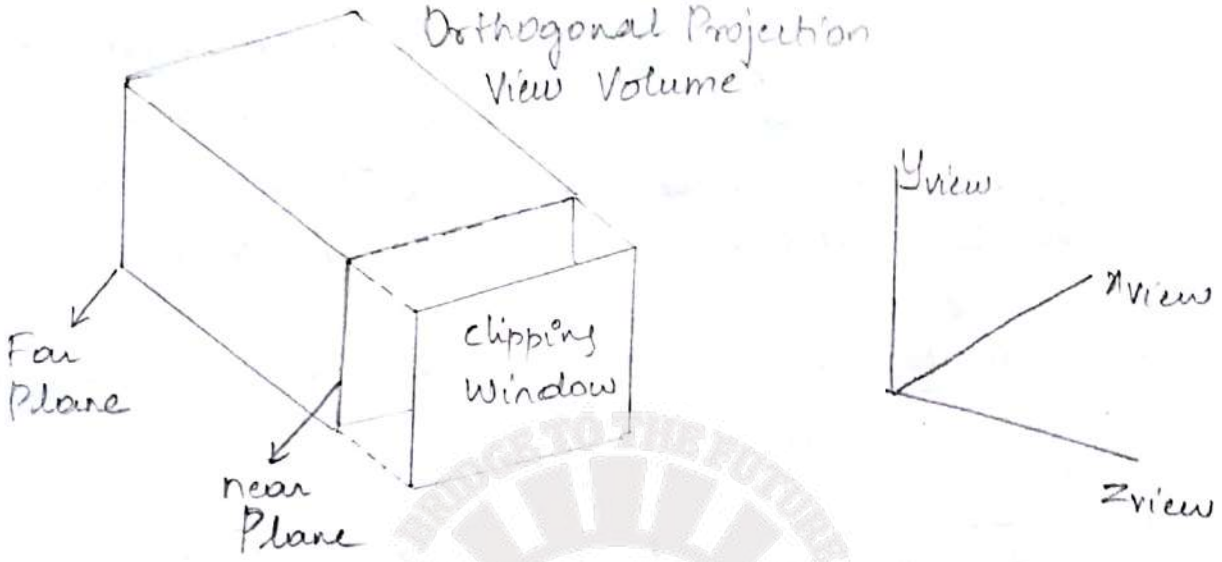




Side View



Top View



Normalization Transformation for an Orthogonal Projection

* Using an Orthogonal transfer of co-ordinate position onto the view plane, projected position of any spatial point (x, y, z) is represented simply as (x, y).

* once the limits of view volume is established, the co-ordinate descriptions inside this rectangular parallelepiped are the projection co-ordinates & they are mapped into a normalized view volume.

* x, y, z co-ordinates are normalized in range from 0 to 1. / range from -1 to 1. represented in left-handed system.

* Position (x_{min}, y_{min}, z_{near} is mapped to normalized position (-1,-1,-1) & position (x_{max}, y_{max}, z_{far}) is mapped to (1,1,1)

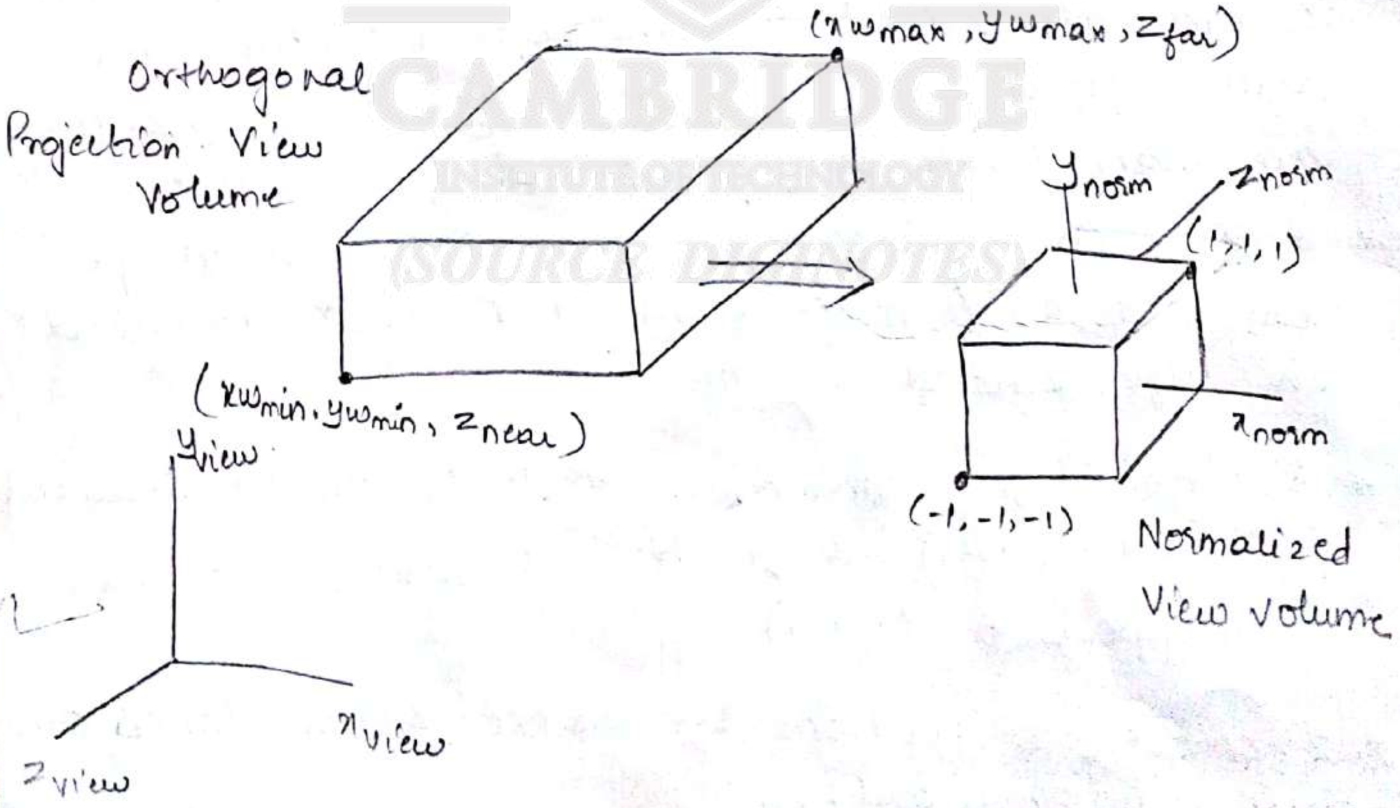
* z-co-ordinate positions for the near & far planes are denoted as z_{near} and z_{far}.

Prof. Supriya S

- Check Module 3 for converting the clipping window into normalized symmetric square. In addition, z-co-ordinate values are to be transformed in the range from z near to zfar. (-1 to 1) using similar calculations as discussed prior.

- The Normalizedation transformation for the orthogonal view volume is

$$M_{ortho,norm} = \begin{bmatrix} \frac{2}{xw_{max} - xw_{min}} & 0 & 0 & \frac{-xw_{max} + xw_{min}}{xw_{max} - xw_{min}} \\ 0 & \frac{2}{yw_{max} - yw_{min}} & 0 & \frac{-yw_{max} + yw_{min}}{yw_{max} - yw_{min}} \\ 0 & 0 & \frac{-2}{z_{near} - z_{far}} & \frac{z_{near} + z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Perspective Projection

- Distance from COP to projection plane is finite. The projectors are not parallel & we specify the center of projection (COP)

COP is also called as Perspective reference point or Projection Reference point.

Perspective foreshortening

The size of the perspective projection of the object varies inversely with the distance of the object from the center of projection.

Vanishing Point:

The perspective projections of any set of parallel lines that are not parallel to the projection plane converge to a vanishing point.

* Perspective projection → projections of distant objects are smaller than the projections of objects of same size that are closer to the view plane.

* The projection path of a spatial position (x, y, z) to a general ~~position~~ projection reference point at $(x_{prp}, y_{prp}, z_{prp})$. The projection line intersects the viewplane at co-ordinate position (x_p, y_p, z_{vp}) where z_{vp} is some selected position for the view plane along the z_{view}-axis.

P(x', y', z') any point along the projection line

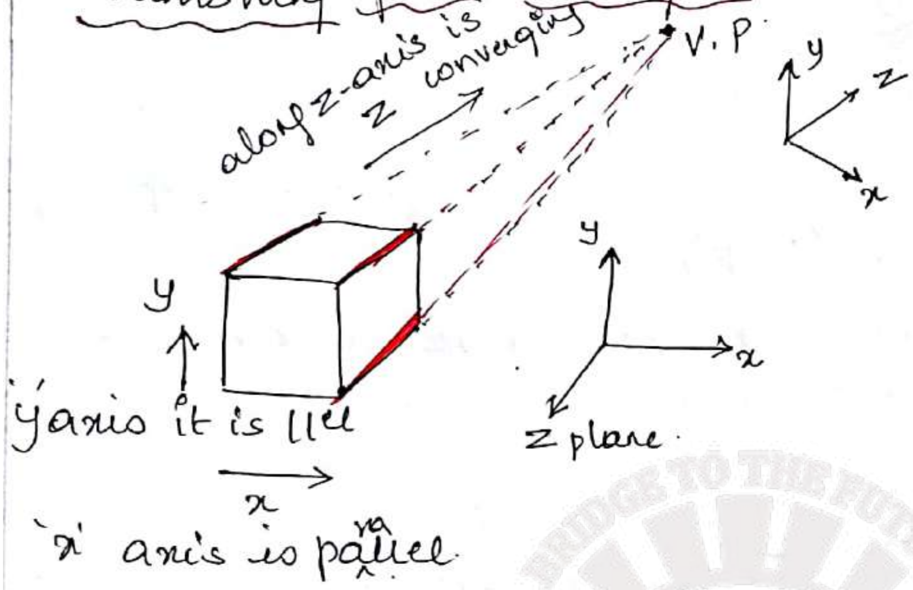
$$\left\{\begin{aligned} x' &= x - (x - x_{prp})u \\ y' &= y - (y - y_{prp})u \\ z' &= z - (z - z_{prp})u \end{aligned}\right\}$$

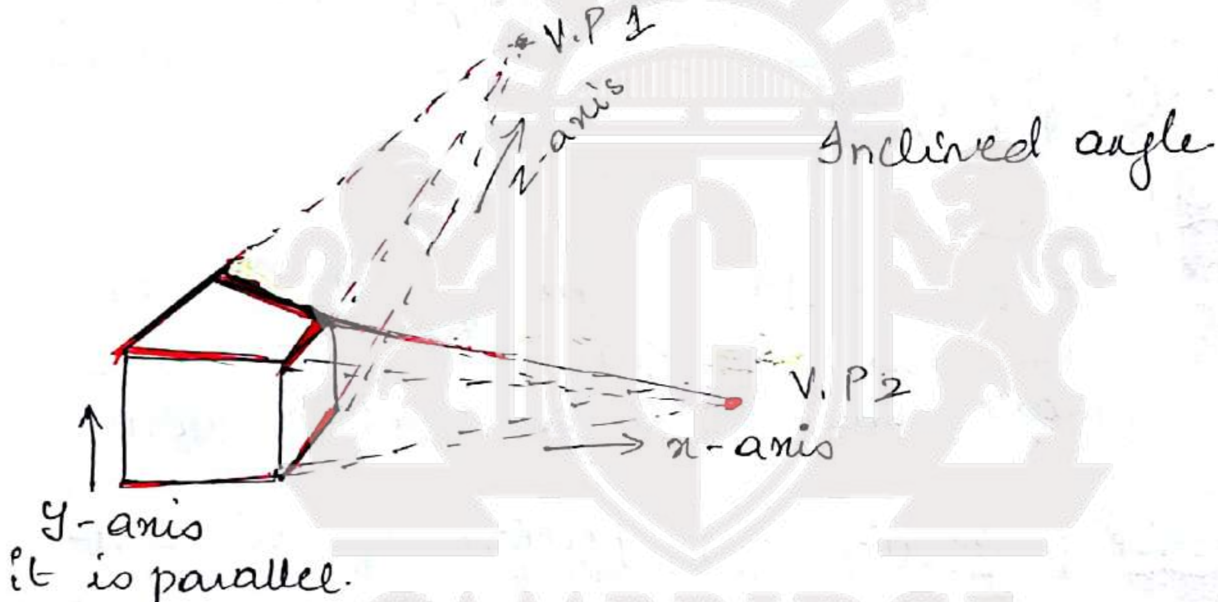
Parametric form

$0 \leq u \leq 1$ explained in module 5

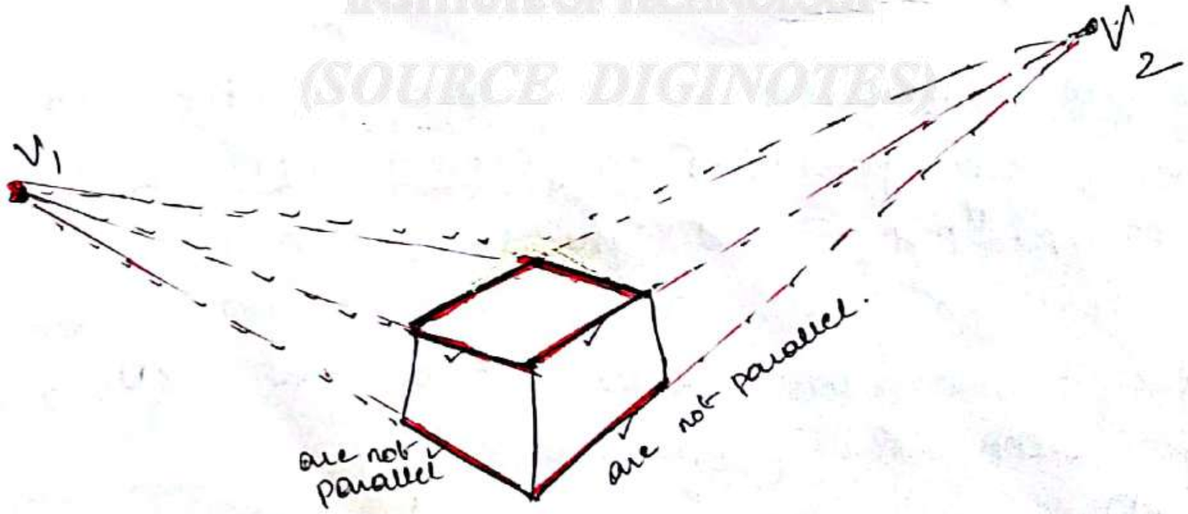
Prof. Supriya S

Vanishing point example.





(OR)



For a set of points, that are parallel to one of the principle axis of an object, it is refered to as Principal vanishing point

Cont^n:

* where $x' y' z'$ represents any point along the projection line.

$u = 0$, when we are at point $P = (x, y, z)$

$u = 1$, when we are at the other end $(x_{prp}, y_{prp}, z_{prp})$

* on view plane $z' = z_{vp}$, solving z' for parameter 'u' at position $z' = z_{vp}$ along the projection line

$$z' = z - (z - z_{prp})u$$

$$z_{vp} = z - (z - z_{prp})u$$

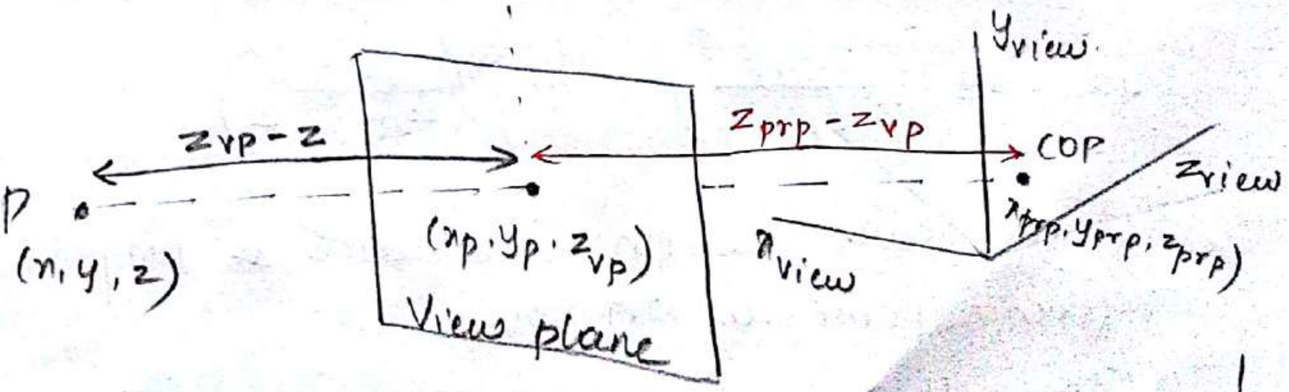
$$(z_{vp} - z) = -(z - z_{prp})u$$

$$u = \frac{z_{vp} - z}{z_{prp} - z}$$

$$\boxed{u = \frac{z_{vp} - z}{z_{prp} - z}}$$

Substituting 'u' in x' & y', we get general perspective transformation eq^n's.

$$\boxed{\begin{aligned} x_p &= x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + x_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right) \\ y_p &= y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + y_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right) \end{aligned}}$$



$$COP + u(P - COP) \quad 0 \leq u \leq 1$$

(OR)

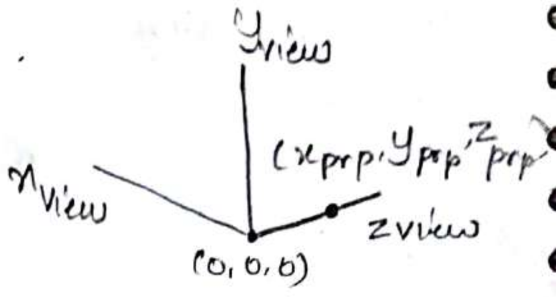
$$P + u(P - COP)$$

Prof. Supriya S

Perspective - Projection Equations : Special cases.

The projection reference point could be limited to positions along z-axis then.

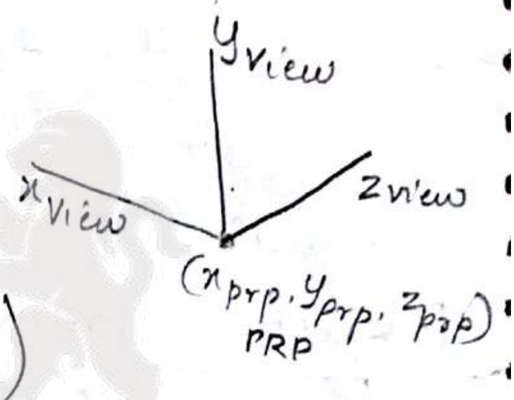
① $x_{prp} = y_{prp} = 0$



then
$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right)$$

$$y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right)$$

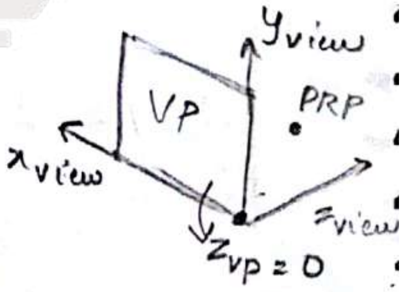
② $(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 0)$



$$x_p = x\left(\frac{z_{vp}}{z}\right) \quad y_p = y\left(\frac{z_{vp}}{z}\right)$$

③ If view plane is on the UV plane.

$z_{vp} = 0$:



$$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right) - x_{prp}\left(\frac{z}{z_{prp} - z}\right)$$

$$y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right) - y_{prp}\left(\frac{z}{z_{prp} - z}\right)$$

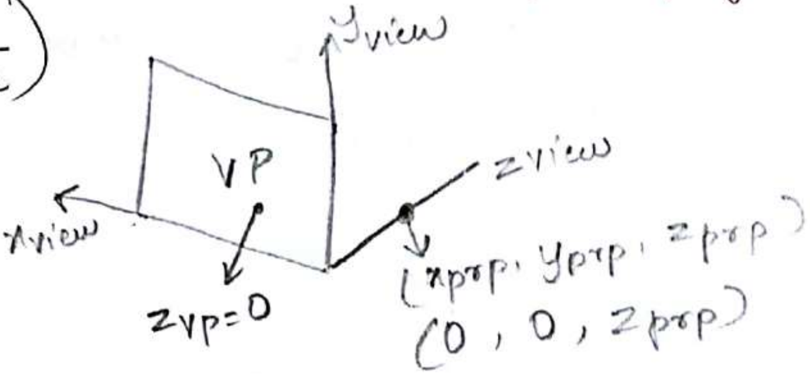
④ if uv plane is on viewplane & projection reference point is on z_{view}.

$x_{prp} = y_{prp} = z_{vp} = 0$

then

$$x_p = x\left(\frac{z_{prp}}{z_{prp}-z}\right)$$

$$y_p = y\left(\frac{z_{prp}}{z_{prp}-z}\right)$$

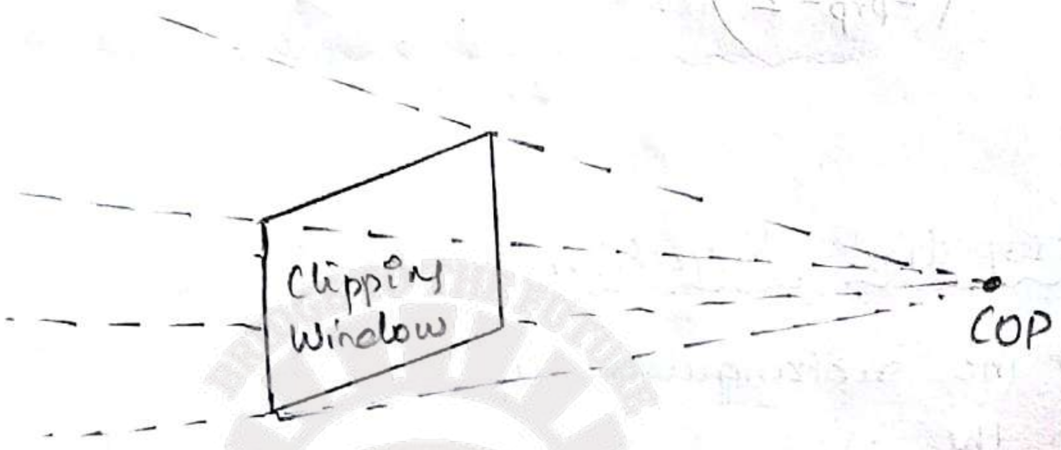


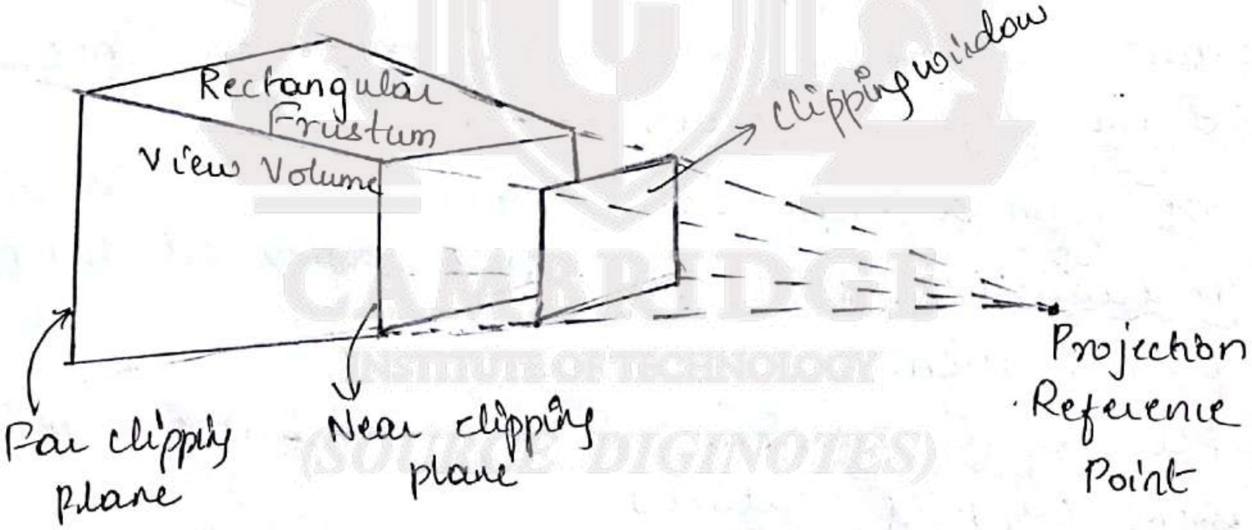
Perspective - Projection View Volume.

- The rectangular clipping window is positioned on the view plane.
- The bounding planes ~~of~~ for the view volume are not parallel, because the projection lines are not parallel.
- The bottom, top, & sides of view volume are planes through window edges that all intersect at the projection reference point.
- It form a view volume that is an infinite rectangular pyramid with its apex at the center of projection.
- All objects outside this pyramid are eliminated using clipping routines.
- The perspective projection view volume is refered as pyramid of vision. (cone of vision of our eyes or camera).
- By adding near and far clipping planes that are perpendicular to Zview axis (& ||el to view plane , the other parts of the infinite perspective projection view.

Prof. Supriya S

volume are chopped off, forming a truncated pyramid or frustum, view volume.





Prof. Supriya S

Perspective Projection transformation matrix

perspective projection equations.

$$x_p = x\frac{(z_{prp}-z_{vp})}{(z_{prp}-z)} + x_{prp}\left(\frac{z_{vp}-z}{z_{prp}-z}\right)$$

$$y_p = y\left(\frac{z_{prp}-z_{vp}}{z_{prp}-z}\right) + y_{prp}\left(\frac{z_{vp}-z}{z_{prp}-z}\right)$$

consider the denominator

$h = z_{prp} - z$

and also consider

$$x_h = x(z_{prp}-z_{vp}) + x_{prp}(z_{vp}-z)$$

$$y_h = y(z_{prp}-z_{vp}) + y_{prp}(z_{vp}-z)$$

Place all the x, y, z corresponding values in the matrix form

using homogeneous co-ordinate representations, we get

$$x_p = \frac{x_h}{h} \quad \text{and} \quad y_p = \frac{y_h}{h}$$

The perspective-projection transformation of a viewing co-ordinate position is then accomplished in 2 steps

① calculate homogenous co-ordinates using the perspective transformation matrix.

$$\boxed{P_h = M_{pers} \cdot P}$$

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ 1 \end{bmatrix} = \begin{array}{cccc} x & y & z & h \\ \left[\begin{matrix} z_{prp}-z_{vp} \\ 0 \\ 0 \\ 0 \end{matrix}\right. & \begin{matrix} 0 \\ z_{prp}-z_{vp} \\ 0 \\ 0 \end{matrix} & \begin{matrix} -x_{prp} \\ -y_{prp} \\ s_z \\ -1 \end{matrix} & \left.\begin{matrix} x_{prp}z_{vp} \\ y_{prp}z_{vp} \\ t_z \\ z_{prp} \end{matrix}\right] \end{array} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

② after the normalized transformation & clipping routines are applied, homogeneous co-ordinates are divided by parameter h to obtain the true transformation - co-ordinate positions.

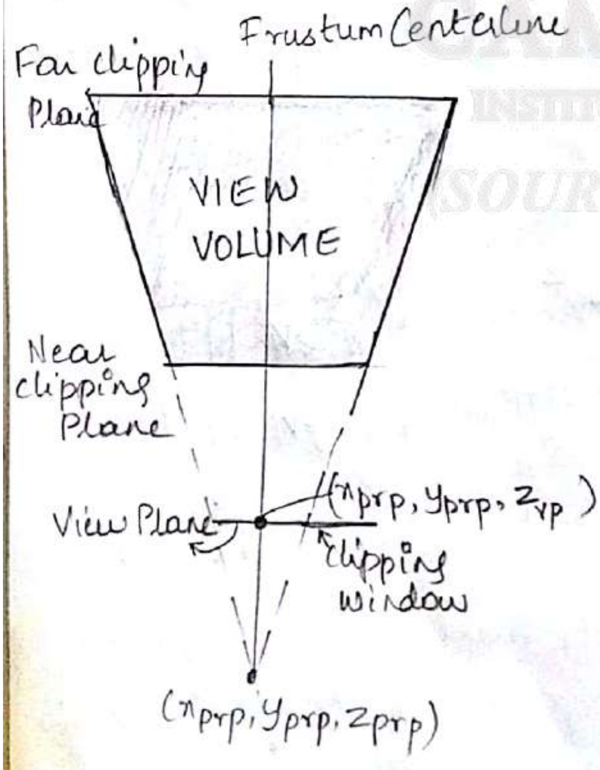
Symmetric Perspective-Projection frustum

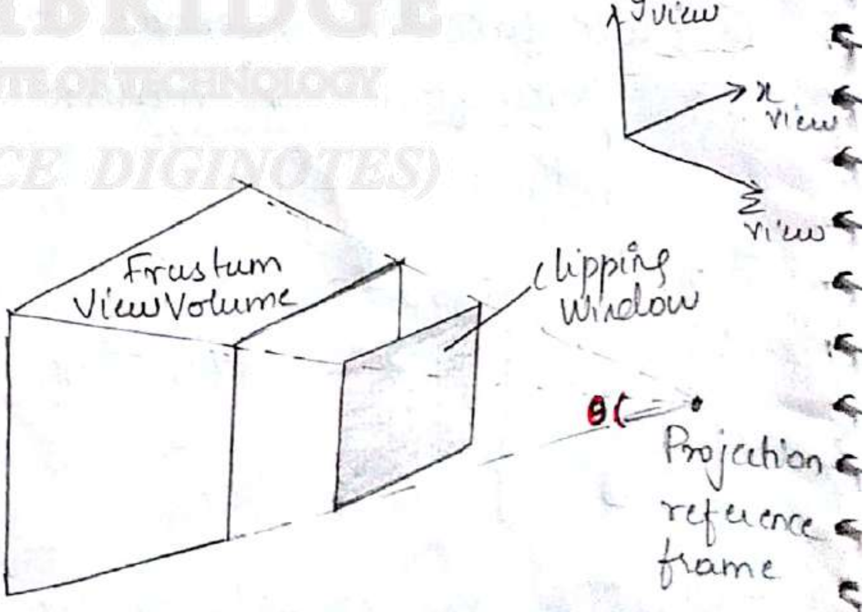
The line from the projection reference point through the center of the clipping window & on through the view volume is the centerline for a perspective projection frustum. If this centerline is perpendicular to view plane - symmetric frustum.

* Frustum centerline intersects the view plane at co-ordinate location $(x_{prp}, y_{prp}, z_{vp})$. The corner positions of clipping window are

$$x_{wmin} = x_{prp} - \frac{width}{2} \qquad x_{wmax} = x_{prp} + \frac{width}{2}$$

$$y_{wmin} = y_{prp} - \frac{height}{2} \qquad y_{wmax} = y_{prp} + \frac{height}{2}$$



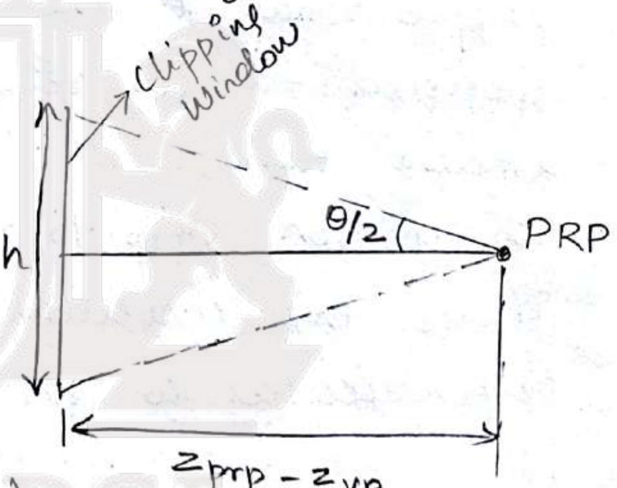


- Another way to specify a symetric perspective projection is to use parameters / ~~of~~ properties of camera lens.

A photograph is produced with symetric PP of a scene onto a film plane. Reflected light rays from the objects in scene are collected on film plane from within the "cone of vision" of the camera. This cone of vision is referenced with field-of view angle – measure of size of camera lens.

- field-of view angle → angle b/w the top clipping plane and bottom clipping plane of frustum (determines the height)

aspect ratio $= \frac{\text{width}}{\text{height}}$



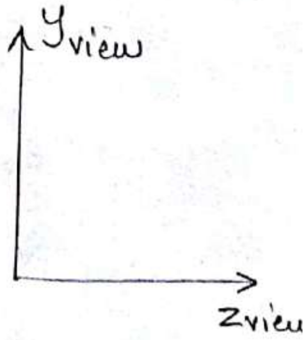
$$\boxed{\tan\left(\frac{\theta}{2}\right) = \frac{\text{height}/2}{z_{prp} - z_{vp}}}$$

$$\text{height} = 2(z_{prp} - z_{vp})\tan\left(\frac{\theta}{2}\right)$$

$z_{prp} - z_{vp}$ can be expressed as

$$z_{prp} - z_{vp} = \frac{\text{height}}{2}\cot\left(\frac{\theta}{2}\right)$$

$$= \frac{\text{width}\cdot\cot(\theta/2)}{2.\text{aspect}}$$



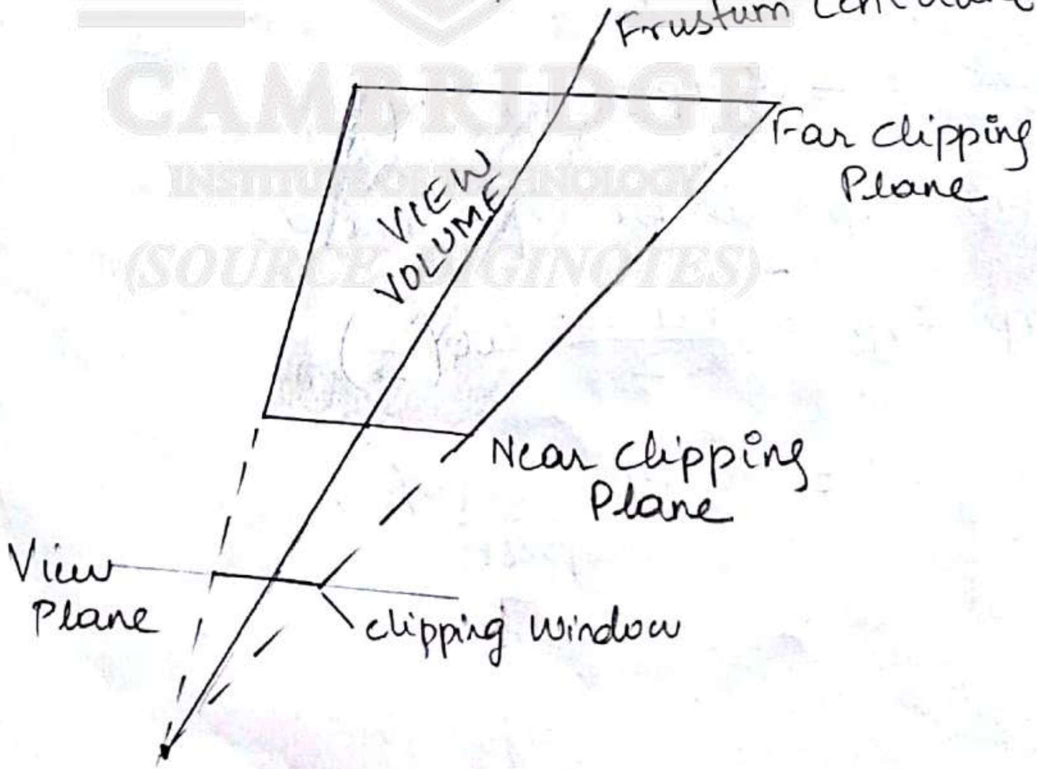
Prof. Supriya S

Oblique Perspective - Projection Frustum

- If the centerline of a perspective-projection view volume is not perpendicular to the view plane: – oblique Frustum

- An oblique PP view volume can be converted to a symmetric frustum by applyiy z-axis shearing transformation matrix.
- It shifts all position on any plane that is perpendicular to the z-axis by an amount that is proportional to the distance of the plane from a specified z-axis reference position. i.e shift by an amount that will move the center of clipping window to position (x_{prp}, y_{prp}) on the view plane.

Hence the centerline is adjusted such that it is perpendicular to the view plane.



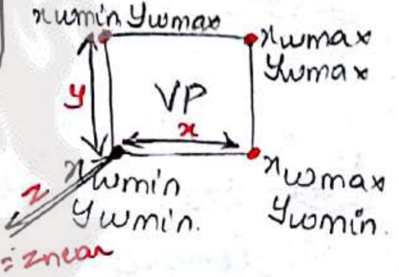
Prof. Supriya S

Taking PRP as $(x_{prp}, y_{prp}, z_{prp}) = (0,0,0)$, Shearing matrix is

$$M_{zshear} = \begin{bmatrix} 1 & 0 & sh_{zx} & 0 \\ 0 & 1 & sh_{zy} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

if the view plane is at the position of the near clipping plane (move it) then $z_{vp} = z_{near}$ & $x_p, y_p = (0,0)$

$$\begin{array}{l} x_p \rightarrow \\ y_p \rightarrow \\ z_{vp} \rightarrow \\ 1 \rightarrow \end{array} \begin{bmatrix} 0 \\ 0 \\ z_{near} \\ 1 \end{bmatrix} = M_{zshear} \cdot \begin{bmatrix} \frac{x_{wmin}+x_{wmax}}{2} \\ \frac{y_{wmin}+y_{wmax}}{2} \\ z_{near} \\ 1 \end{bmatrix} \begin{array}{l} \rightarrow x \text{ direction of view plane} \\ \rightarrow y \text{ direction of view plane} \\ \rightarrow z \text{ direction} \\ \ \end{array}$$



↓ Matrix Multiply

$$\frac{x_{wmin}+x_{wmax}}{2} + sh_{zx} \cdot z_{near}$$

$$\boxed{sh_{zx} = -\frac{(x_{wmin}+x_{wmax})}{2 \cdot z_{near}}}$$

$$\frac{y_{wmin}+y_{wmax}}{2} + sh_{zy} \cdot z_{near}$$

$$\boxed{sh_{zy} = -\frac{(y_{wmin}+y_{wmax})}{2 \cdot z_{near}}}$$

Ill^{ly} the Perspective projection matrix when PRP is at viewing origin $(x_{prp}, y_{prp}, z_{prp}) = (0,0,0)$ and view plane is the near clipping plane $z_{\overset{VP}{\cancel{prp}}} = z_{near}$ then the matrix M_{pres} is simplified as

$$M_{pres} = \begin{bmatrix} -z_{near} & 0 & 0 & 0 \\ 0 & -z_{near} & 0 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

finally

$$\boxed{M_{obliquepers} = M_{pres} \cdot M_{zshear}}$$

substitute for Sh_{zx} & Sh_{zy}

$$= \begin{bmatrix} -z_{near} & 0 & \frac{x_{wmin} + x_{wmax}}{2} & 0 \\ 0 & -z_{near} & \frac{y_{wmin} + y_{wmax}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Normalized Perspective - Projection Transformation Co-ordinates.

The final step in the perspective- projection transformation process is to map ~~to~~ this parallelepiped to a Normalized view volume.

Normalisation matrix for Perspective projection transformation

$$M_{normpers} = M_{xyscale} \cdot M_{obliquepers}$$

$$= \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} -z_{near} & 0 & \frac{x_{wmin} + x_{wmax}}{2} & 0 \\ 0 & -z_{near} & \frac{y_{wmin} + y_{wmax}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} -z_{near}s_x & 0 & s_x\frac{xw_{min}+xw_{max}}{2} & 0 \\ 0 & -z_{near}s_y & s_y\frac{yw_{min}+yw_{max}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

we obtain homogenous co-ordinates

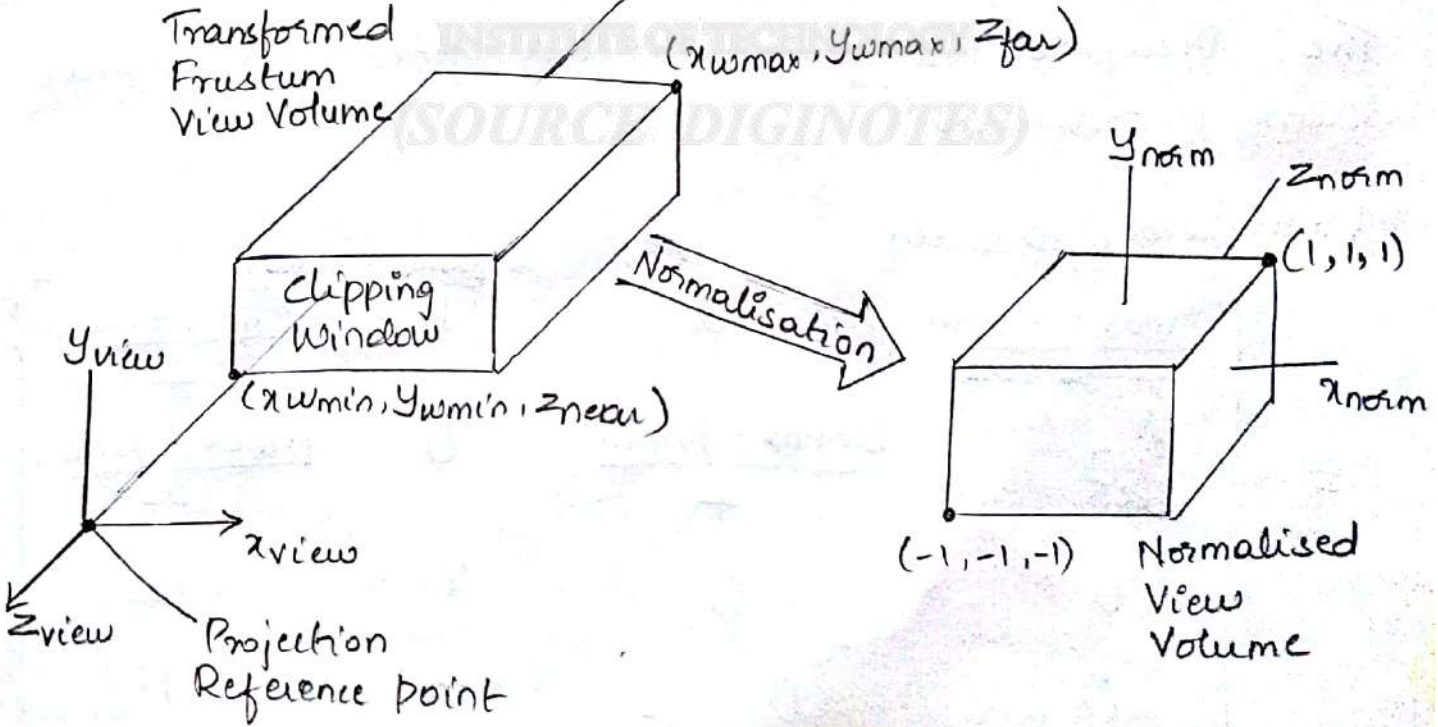
$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \underbrace{M_{normpers} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}$$

and projection coordinates are (multiply the above matrix)

$$x_p = \frac{x_h}{h} = \frac{-z_{near}s_x x + [s_x(xw_{min}+xw_{max})/2]}{-z}$$

$$y_p = \frac{y_h}{h} = \frac{-z_{near}s_y y + [s_y(yw_{min}+yw_{max})/2]}{-z}$$

$$z_p = \frac{z_h}{h} = \frac{s_z z + t_z}{-z}$$



Substitute $(x_{wmin}, y_{wmin}, z_{near})$ as $(-1,-1,-1)$ & $x_{wmax}, y_{wmax}, z_{far}$ as $(1,1,1)$.

$$S_x = \frac{2}{x_{wmax} - x_{wmin}} \qquad S_y = \frac{2}{y_{wmax} - y_{wmin}}$$

$$S_z = \frac{z_{near} + z_{far}}{z_{near} - z_{far}} \qquad t_z = \frac{2 z_{near} z_{far}}{z_{near} - z_{far}}$$

Hence

$$M_{normpers} = \begin{bmatrix} \frac{-2 z_{near}}{x_{wmax} - x_{wmin}} & 0 & \frac{x_{wmax} + x_{wmin}}{x_{wmax} - x_{wmin}} & 0 \\ 0 & \frac{-2 z_{near}}{y_{wmax} - y_{wmin}} & \frac{y_{wmax} + y_{wmin}}{y_{wmax} - y_{wmin}} & 0 \\ 0 & 0 & \frac{z_{near} + z_{far}}{z_{near} - z_{far}} & \frac{2 z_{near} z_{fa}}{z_{near} - z_{fa}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

The Viewport Transformation and three-Dimensional Screen co-ordinates.

$M_{normviewvol, 3D screen}$

$$= \begin{bmatrix} \frac{x_{vmax} - x_{vmin}}{2} & 0 & 0 & \frac{x_{vmax} + x_{vmin}}{2} \\ 0 & \frac{y_{vmax} - y_{vmin}}{2} & 0 & \frac{y_{vmax} + y_{vmin}}{2} \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Prof. Supriya . S

Open GL 3D viewing functions – Refer from text book.

Prof. Supriya S

Visible Surface detection [Hidden Surface elimination]

Given a set of 3D surfaces to be projected onto a 2D screen, obtain the nearest surface corresponding to any point on the screen.

* Some methods require more memory, some involve more processing time, some apply only to special types of objects
* Identifying the surface in 3D requires calculation of 'z' co-ordinate (depth) value & Surface Normal (if curved).

Classification of Visible surface detection Algorithms

① Object space methods (continous)

* compares objects and parts of objects to each other to determine which surfaces should be labeled as visible. (use of bounding box, check limits along each direction).
* Order the surface being drawn, such that it provides the correct impression of depth variations and positions.

② Image space methods (discrete)

Visibility is decided point by point at each pixel position on the projection plane. Screen resolution can be a limitation.

Hidden surfaces – (a) surface for rendering
or
(b) Line drawing.

X

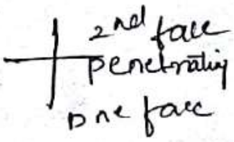
Prof. Supriya . S

Visible surface detection Algorithm most use Sorting and coherence methods to improve Performance.

\# Sorting - is used to facilitate depth comparisions by ordering the individual surfaces in scene according to their distances from the view plane.

\# Coherence methods are used to take the advantage of regularities in the scene

- if one object is entirely seperate from another, do not compare (object coherence)
- face coherence : smooth variations across a face, incrementally modify.
- Edge coherence : Visibility change if a edge cross behind a visible face.
- Implied edge coherence : line of intersection of a planar face penetrating another, can be obtained from two points on the intersection.



- Scan line coherence - successive lines have similar spans.
- Depth coherence - use difference equation to estimate depths of nearby points on the same surface.
- Frame coherence - Pictures of 2 successive frames of an animation sequence are quite similar. (small changes ~~of~~ in object & viewpoint).

Prof. Supriya S.

Back-face detection (Object-space Method)

A polygon (in 3D) is a back face if $V.N > 0$

- * concepts of front-back tests
- * A point (x,y,z) is behind a polygon surface if

$$Ax + By + Cz + D < 0$$

where A, B, C, D are plane parameters for the polygon

- * consider the direction of normal vector N for a polygon surface, if V_{view} is a vector in the viewing direction from our camera position., the polygon is back face if $\boxed{V_{view} . N > 0}$

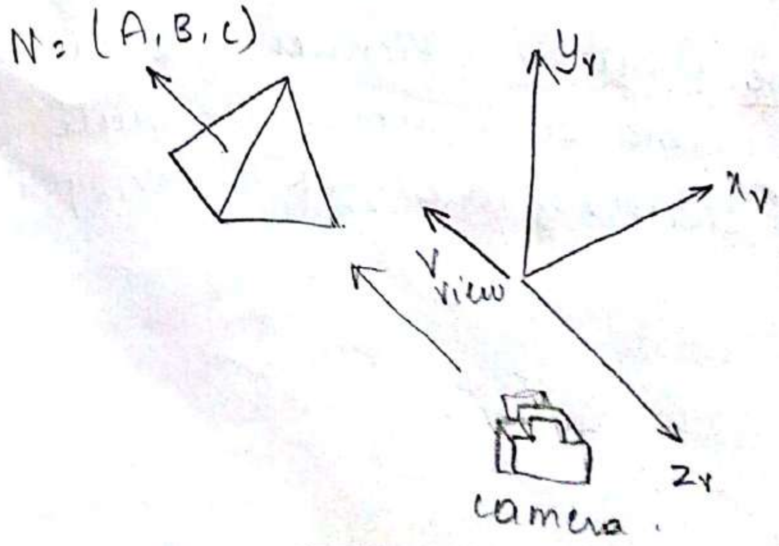
(viewing direction is parallel to viewing V_z axis)

Let $V = (0,0,V_z)$ and $N = Ax + By + Cz$.

the $V.N = V_z . \underline{C}$ (z component of normal vector N is to be considered)

if the viewing direction is along the -ve z-axis, a polygon is a back face if the z component, C, of its normal N satisfies $C<0$.

$\therefore \boxed{C \leq 0}$

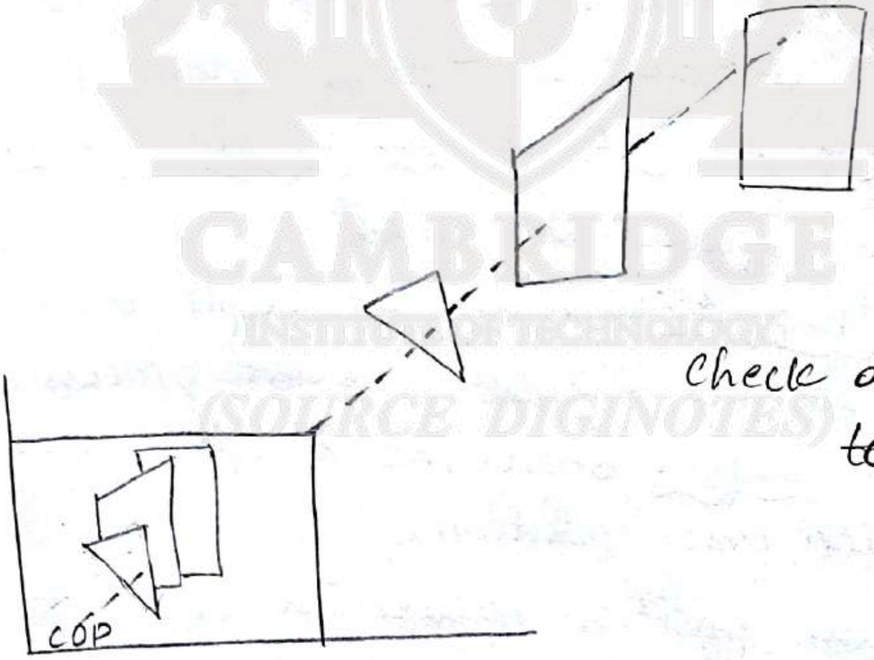


Prof. Supriya S

- Back face have normal vectors that point away from the viewing position (viewing direction is along +ve z-axis). are identified by $\boxed{C \geq 0}$.
- For convex polygons, this test identifies all the hidden surfaces in scene since each surface is completely visible or completely hidden.
- For concave polygons, more tests are to carried out to determine whether there are any additional faces that are totally or partially obscured by other faces.

Depth - Buffer Method. (image space approach)

also referred as z-buffer method.



Check algorithm from text Book

- It compares surface depth values throughout a scene for each pixel position on the projection Plane.
- Each surface of scene is processed seperately, one

Prof. Supriya S

pixel at a time across the surface.

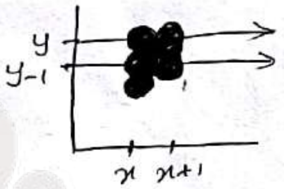
- Refered to as z-buffer since object depth is usually measured along the z-axis of viewing system.
- The figure shows 3 surfaces at varying distances along the orthographic projection line from position (x,y) on a view plane.
- Surfaces can be processed in any order
- As each surface is processed, its depth from the view plane is compared to previously processed surface.
- If a surface is closer than any previously processed surfaces, its surface color is calculated & saved, along with its depth.
- depth ~~colo~~ buffer algorithm is typically carried out in normalized co-ordinates - range from 0 at near clipping plane to 1.0 at far clipping plane.
- The method requires 2 buffers
  - depth buffer used to store depth values for each (x,y) position as surfaces are processed.
  - frame buffer stores the surface-color values for each pixel position.
- Initially all position values in depth buffer are set to 1.0 (maximum depth), frame buffer is initialised to the background color.
- Each surface listed in polygon tables are processed, one scan line at a time, by calculating the depth

values at each (x,y) pixel position.

- This calculated depth is compared to the value previously stored in depth buffer for that pixel position
- If the calculated depth is less than the value stored in the depth buffer, a new depth value is stored.
- Then the surface color ~~of~~ at that position is computed & placed in the corresponding pixel location in the frame buffer.

At surface position (x,y) the depth is calculated from the plane equation as

$$z = \frac{-Ax - By - D}{C}.$$



If x values along the horizontal lines differ by ±1 & y values ~~along~~ on adjacent scan lines differ by ±1

If the depth of position (x,y) has been determined to be z, then the depth z' of next position (x+1, y) is

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

$$= \frac{-Ax - By - D}{C} \mp \frac{A}{C}.$$

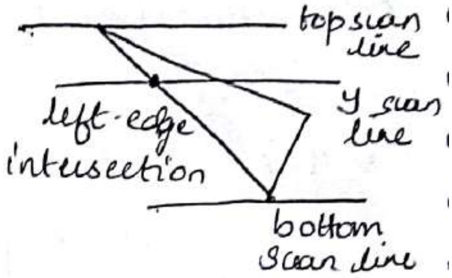
$$\boxed{z' = z \mp \frac{A}{C}}$$

$-A/C$ is constant for each surface.

→ Successive depth values across a scanline are obtained by just adding with single value addition $(-A/c)$

We can Prof. Supriya S

- To implement the depth buffer algorithm by starting at a top vertex of the polygon, we could then calculate recursively the x-value down the left edge of the polygon.



using slope of equation

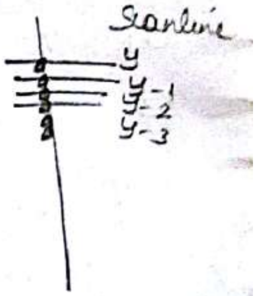
$$x' = x - \frac{1}{m} \quad , y = y-1$$

$$\therefore \quad z' = \frac{-A(x - 1/m) - B(y-1) - D}{C}$$

$$= \frac{-Ax - By - D}{C} + \frac{A/m + B}{C}$$

$$\boxed{z' = z + \frac{A/m + B}{C}}$$

→ depth values down the edge are obtained



- proceeding down the vertical edge

$$x = x \quad , \quad y = y-1$$

$$z' = \frac{-Ax - B(y-1) - D}{C}$$

$$= \frac{-Ax - By - D}{C} + \frac{B}{C}$$

$$\boxed{z' = z + \frac{B}{C}}$$

Prof. Supriya S

OpenGL Visible Surface-detection functions / visibility detection functions.

① OpenGL Polygon Culling functions.

Back face removal is accomplished with

- glEnable (GL-CULL-FACE);
- glCullFace (mode);

mode is assigned to GL-BACK (default) :
GL-FRONT
GL-FRONT-AND-BACK

- glDisable (GL-CULL-FACE);

② OpenGL Depth Buffer functions.

- glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

Depth Buffer values can then be initialized with

- glClear(GL_DEPTH_BUFFER_BIT);

There is no need to clear the depth buffer each time we want to display a new frame. In OpenGL depth values are normalized in the range from 0 to 1.0, so that the preceding initialization sets all the depth buffer values to max of 1.0 by default.

Depth buffer visibility detection routines are activated using glEnable (GL_DEPTH_TEST);
glDisable (GL-DEPTH_TEST);

Prof. Supriya CS

Q. Depth-buffer visibility testing can also be set with other initial value for the maximum depth.

glClearDepth (maxDepth);

↓ can be set to any value between 0 & 1.

It loads this initialization value into depth buffer.

next glClear (GL_DEPTH_BUFFER_BIT) must be invoked.

* Projection co-ordinates are normalized to range from -1.0 to 1.0

depth values b/w near & far clipping planes are normalised to range from 0.0 to 1.0.

Hence we can adjust these normalization values with

glDepthRange (nearNormDepth, farNormDepth);

↓ 0.0 ↓ 1.0 by default.

* glDepthFunc(testCondition);

↳ GL_LESS (default)

GL_GREATER

GL_EQUAL

GL_NOTEQUAL

GL_LEQUAL

GL_GEQUAL

GL_NEVER.

GL_ALAWAYS.

Prof. Supriya S

To check if the depthbuffer status is read-only or read-write status.

glDepthMask (writeStatus)

↳ GL_TRUE (default value) ← both read & write

GL_FALSE (write mode for depth buffer is disabled)

③ OpenGL ~~[illegible]~~ wire-frame surface visibility methods

glPolygonMode (GL_FRONT_AND_BACK , GL_LINE)

④ OpenGL depth-cueing functions.

To vary the brightness of an object as a function of its distances from the viewing position with .

glEnable (GL_FOG);

glFogi (GL_FOG_MODE , GL_LINEAR);

It applies linear depth fuction values to object colors using $d_{min} = 0.0$ & $d_{max} = 1.0$.

dmin & dmax can be set to different values.

glFogf (GL_FOG_START , minDepth);

glFogf (GL_FOG_END , maxDepth);

minDepth & maxDepth → assigned floating point values.

Prof. Supriya S

MODULE - 5

Curve Representation

Non-parametric form

$y = f(x)$

↓

Implicit form

$f(x,y) = 0$ → bresenham's clipping.

Explicit form.

$y = mx + b$

↓

clipping?

Parametric form.

$x = x(t)$

$y = y(t)$

non-planar objects - spheres, ellipse, cones (primitive of OpenGL).

Some applications observe drawbacks with ~~plothar~~ Polygon patches

① Sharp bends in 2D/3D

② Large changes of curvatures of surface

③ Gradient changes very fast.

Intersection of 2 surfaces - yields a curve

Prof. Supriya S

Programming Event-Driven Input

① Using the pointing Device

2 types of events are associated with pointing devices.

- A mouse Event is generated when a mouse is moved with one of the buttons pressed (or released)
- If a mouse is moved without a button being held down, this event is called Passive move event.
- The information returned includes the button that generated the event, the state of button after the event (up or down) & the position of cursor tracking the mouse in window co-ordinates.
- Register the mouse callback function in Main()
  glutMouseFunc(myMouse);
- mouse callback must have the form

```
void myMouse(int button, int state, int x, int y)
{
    If (button == GLUT-LEFT-BUTTON && state ==
    GLUT_DOWN)
        exit(0);
}
```

- Reshape event is generated whenever the window is resized (by user interaction)

② Window Event.

Prof. Supriya S

- allows the user to resize the window interactively, usually by using the mouse to drag a corner of the window to a new location → Window Event.

- If the window size changes, 3 questions are to be considered

① Do we redraw all the objects that were in the window before it was resized?

② What if the aspect ratio of new window is different from the old window.

③ Do the size or attributes of new primitives changes if the size of new window is different from that of old.

```
void MyReshape(int w, int h)
{

}
```

glutReshapeFunc(myReshape) ⟶ Call back function.

- window movement without Resizing.

glutMotionFunc(drawsquare);

③ Keyboard Events.

Keyboard Events are generated when the mouse is in the window and one of the keys is pressed or released.

glutKeyBoardFunc → callback for events generated by pressing the key.

glutKeyboardUpfunc → event generated by releasing a key.

- glutKeyboardFunc (mykey);

- Void mykey (unsigned char key, int x, int y)

```
{ if (key == 'q' || key == 'Q') exit();
}
```

④ The Display & Idle Callbacks

- glutDisplayFunc (myDisplay);

- glutPostRedisplay(); → avoids extra or unnecessary screen drawings by setting a flag inside GLUT's main loop indicating that the display needs to be redrawn.

- Idle callback → continue to generate graphical primitives through a display function while nothing else is happening. It is involved when there are no other events.

⑤ Window Management.

```
id = glutCreateWindow ("Window");
glutSetWindow (id);
```

Prof. Supriya S

Menus

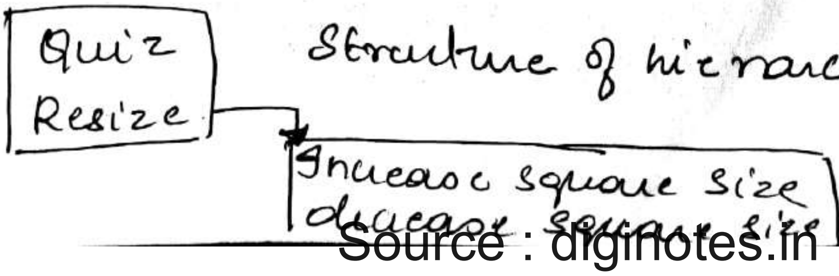
GLUT provides one additional feature, pop-up menus, that would be used with the mouse to create sophisticated interactive applications.

Menus involves

1) define the actions corresponding to each entry in the menu.
2) Link the menu to a particular mouse button
3) finally register a callback function for each menu.

Ex:

```
glutCreateMenu(demo_menu);                        Register callback
glutAddMenuEntry("quit", 1);                      function for eachmenu
glutAddMenuEntry("Increase square size", 2);
glutAddMenuEntry("decrease square size, 3);
glutAttachMenu(GLUT_RIGHT_BUTTON);  → Link
                                        menu to
void  demo_menu(int id)               mouse button.
{   switch(id)
    {  case 1: exit(0);
       break;                          action corresponding
       case 2: size = 2*size;          to each entry in
       break;                          the menu
       case 3: if(size>1) size=size/2;
       break;
    }glutPostRedisplay();
}
```



Structure of hierarchical menu

Prof. Supriya S

The Structure of hierarchical menu is show in the previous page. The code is as follows.

```
sub_menu = glutCreateMenu(size-menu);
glutAddMenuEntry("Increase square size", 2);
glutAddMenuEntry("Decrease square size", 3);
glutCreateMenu(top-menu);
glutAddMenuEntry("Quit", 1);
glutAddSubMenu("Resize", sub_menu);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

Picking

Picking is the logical operation that allows the user to identify an object on the display. Action of picking uses pointing device.

3 ways

1) Selection : involves adjusting the clipping region and viewport, tracks which primitives in a small clipping region are rendered into the region near the cursor. These primitives are placed in the hit list to be examined later.

2) bounding boxes or extents (objects of interest) extent of an object is a smallest rectangle, alligned with co-ordinate axes, that contains the object.

Prof. Supriya S

3) using back buffer & an extra rendering. Double buffering - 2 color buffers. ① A front buffer & back buffer

function glRenderMode (-) can be one of three modes.

① normal rendering to the color buffer (GL-RENDER)

② selection mode (GL-SELECT)

③ feed back mode (GL-FEEDBACK) - used to obtain a list of primitives that were rendered

when selection mode or ~~render~~ ~~a~~ screen is selected, each primitive with ~~in~~ ~~a~~ the clipping window/volume generates a message called a hit that is stored in a buffer called name stack.

glSelectBuffer - used to identify an array for selected data.

i.e 4 important functions for initialising the name stack, for pushing & poping info on it, for manipulating the top entry ~~of~~ on stack.

void glSelectBuffer (GLsizei n , GLuinit * buffer.

void glInitNames () → initialises name stack

void glPushName(GLuint name) - pushes name on the name stack.

void glPopName () - pops the top name from the name stack.

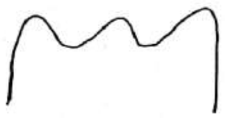
Prof. Supriya S

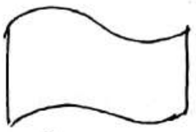
void glLoadName(GLuint name) — replaces top of name stack with name.

gluPickMatrix (x, y, w, h, #vp);

Prof. Supriya S

Curved Surfaces.


curves


Surfaces

- Surface modeling
  free form curves
  and surfaces.

- In Computer graphics, one would be interested to visually represent an object with mathematical representation (Geometric modeling)

Parametric Representation.

Planar eqn

$x = r\cos(t)$ $\qquad$ $x = a+bu+cw$

$y = r\sin(t)$ $\qquad$ $y = d+eu+fw$

$z = h$ $\qquad\qquad$ $z = g+hu+iw$

where every co-ordinate point on a plane (x,y,z) can be represented as a function of 2 variable (u,w) used for surfaces.

- Parametric equations completely seperate the roles dependent & independent variables

- case 1 : $r=5$, $t=\pi$, $z=20$ represents a "point"

- case 2 : $r=5$, $-\pi \leq t \leq \pi$, $z=20$ represents a "circle".

- case 3 : $r=5$, $-\pi \leq t \leq \pi$, $0 \leq z \leq 20$. represents ~~circular~~ cylindrical surface.

- case 4 : vary t & r - circular disk - plana entity

- case 5 : vary r & z - rectangle.

case 6 : $0 \leq r \leq 5$, $-\pi \leq t \leq \pi$, $0 \leq z \leq 20$

Solid cylinder.

* Offers more degree of freedom for controlling the shape of curves & surfaces.

Explicit form

$y = px^3 + qx^2 + rx + s$

Parameteric form

$$\left.\begin{aligned} x &= au^3 + bu^2 + cu + d \\ y &= eu^2 + fu^2 + gu + h \end{aligned}\right\} \text{cubic eq}^n$$

* Relative positions with the other [Transformations] are easy to apply

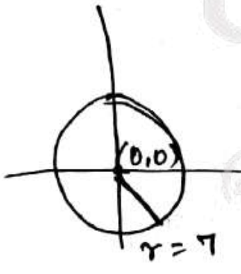
Ex: circle with center (0,0) & radius-7 $\xrightarrow{\text{transformed}}$ circle with center (4,3) & radius-7

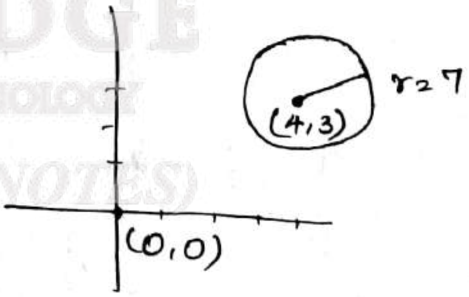
$x = 7\cos(t)$

$y = 7\sin(t)$

$x = 4 + 7\cos(t)$

$y = 3 + 7\sin(t)$





* From the programmer view (Geometric entities)

A point is represented in vector form, transformations are in matrices.

Vectors & matrices

↕ ↑

Geometric entity transformation representations

* Has an advantage in representing the bounds of geometric entities.

circle	circular arc
$x = r\cos(t)$	$x = r\cos(t)$
$y = r\sin(t)$	$y = r\sin(t)$
$r = 8,\ -\pi \le t \le \pi$	$r = 8,\ \underline{-\pi \le t \le 0}$

Quadric Surfaces – frequently used class of objects described with second-degree equations. Includes spheres, ellipsoids, tori, paraboloids, hyperboloids.

Sphere

In Implicit/cartesian co-ordinates, a spherical surface with radius r centered at co-ordinate origin is defined as set of point (x, y, z) that satisfy the eqn.

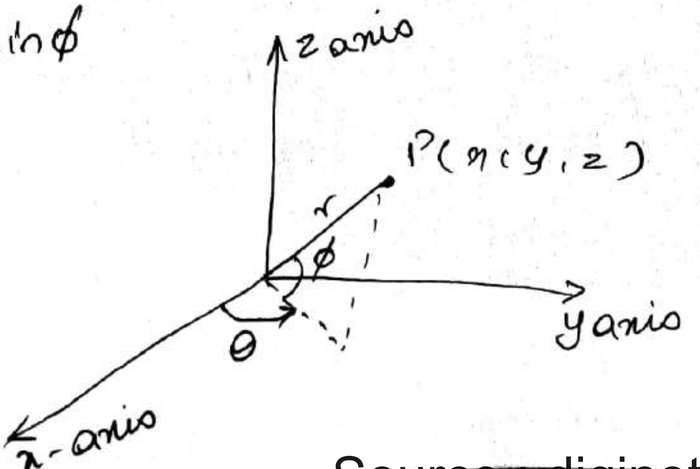
$$x^2+y^2+z^2=r^2$$

Parametric form.

$$x = r\cos\phi\cos\theta \qquad -\pi/2 \le \phi \le \pi/2$$

$$y = r\cos\phi\sin\theta \qquad -\pi \le \theta \le \pi$$

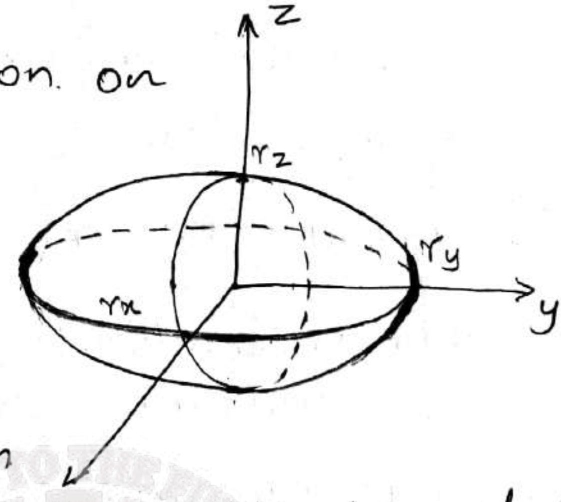
$$z = r\sin\phi$$



Ellipsoid

cartesian representation on the origin

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



Parametric representation for the ellipsoid in terms of latitude angle ϕ & longitude angle θ

$x = r_x \cos\phi \cos\theta \quad -\pi/2 \le \phi \le \pi/2$

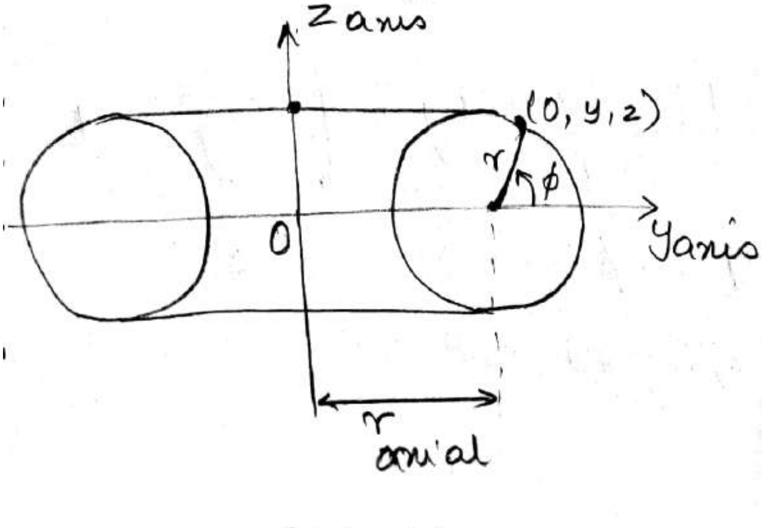
$y = r_y \cos\phi \sin\theta \quad -\pi \le \theta \le \pi$

$z = r_z \sin\phi$

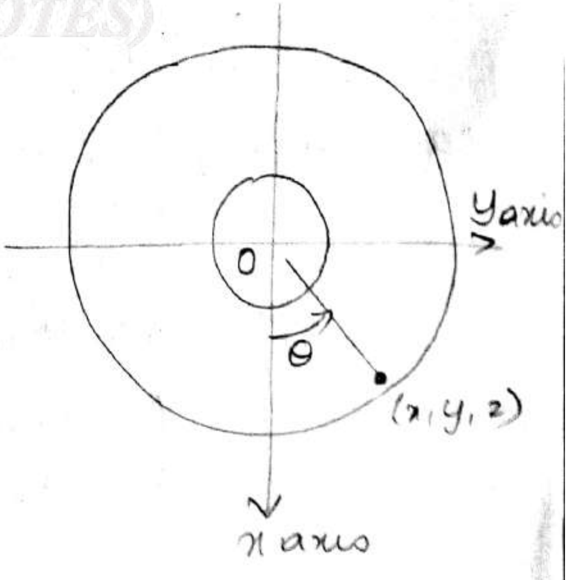
Torus

A doughnut shaped object is called a torus or anchor ring.

parameters for a torus - the distance of conic center from the rotation axis & the dimensions of the conic.



Side View



Prof. Supriya S.

The equation for the cross-sectional circle in side view figure is

$$(y - r_{axial})^2 + z^2 = r^2$$

Rotating this circle about z-axis produces the torus with cartesian equation.

$$(\sqrt{x^2+y^2} - r_{axial})^2 + z^2 = r^2$$

Parametric eq^n for torus with circular cross section are

$$x = (r_{axial} + r\cos\phi)\cos\theta \qquad -\pi \le \phi \le \pi$$
$$y = (r_{axial} + r\cos\phi)\sin\theta \qquad -\pi \le \theta \le \pi$$
$$z = r\sin\phi$$

We can generate a torus by rotating an ellipse instead of a circle. about z-axis

For an ellipse in yz plane with semimajor & semiminor axis denoted by r_y & r_z

$$\left(\frac{y - r_{axial}}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

cartesian eq^n $\Rightarrow$ $\left(\frac{\sqrt{x^2+y^2} - r_{axial}}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$

Parametric eq^n

$$x = (r_{axial} + r_y\cos\phi)\cos\theta \qquad -\pi \le \phi \le \pi$$
$$y = (r_{axial} + r_y\cos\phi)\sin\theta \qquad -\pi \le \theta \le \pi$$
$$z = r_z\sin\phi$$

Prof - Supriya S

Curves can be classified in number of ways

\# Plane curves & space curves

Ex : Circle & Helix → curve is a line (straightness)
is a circle (having equal radius)

\# Curves of known forms & free form curves

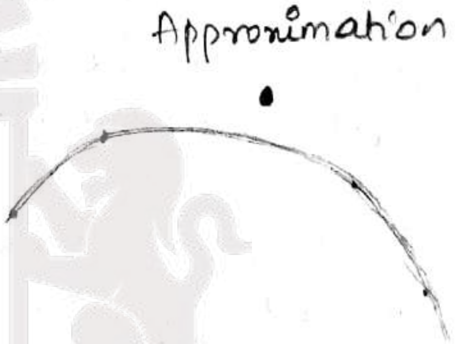
Ex : Circle vr Bezier curve

In order to define a curve, certain conditions are to be satisfied

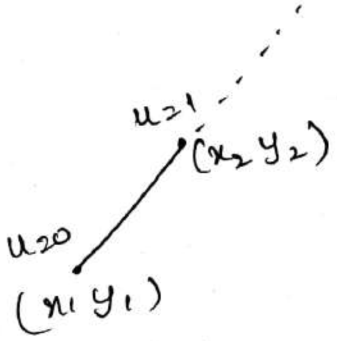
\# Interpolation curves & Approximation curves
↳ If conditions may not be satisfied

Ex : Hermite curve vr Bezier curve

Ex : set of points, curves should pass through all the points



A straight line is a subset of a curve.



Line in any plane

$$\left.\begin{aligned} x &= x_1 + (x_2 - x_1)u \\ y &= y_1 + (y_2 - y_1)u \\ 0 &\le u \le 1.0 \end{aligned}\right\} \text{Parametric representation}$$

To find any point (x, y) at any time interval

$x = x_1 + t_1 \Delta x$ here $\Delta x = x_2 - x_1$

$y = y_1 + t_1 \Delta y$ $\Delta y = y_2 - y_1$

Hence $x = x_1 + (x_2 - x_1)u$

$y = y_1 + (y_2 - y_1)u$

Prof. Supriya S

Spline Representations.

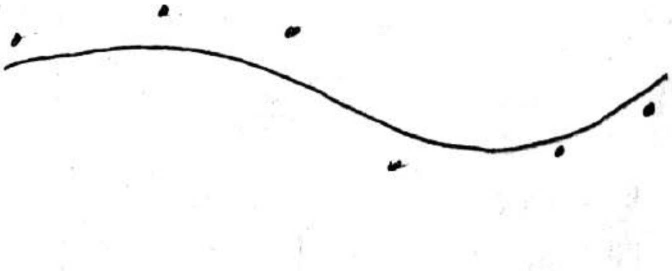
Spline is a flexible strip used to produce a smooth curve through a disignated set of points

* Interpolation and Approximation splines

A spline curve can be specified by giving a set of co-ordinate points, called control points, which Indicate shape of the curve.

A curve generated by connecting all the control points → the resulting curve is said to interpolate the set of control points.

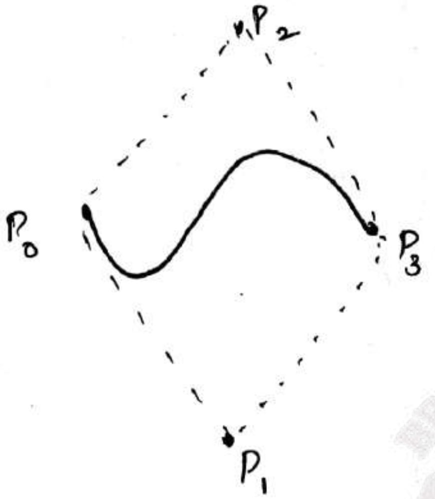
A curve generated by connecting / plotting some or all, of the control points are not on the curve path, the resulting curve is said to approximate set of control points

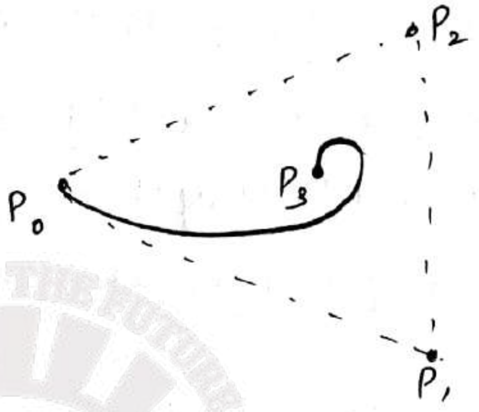


Approximation.

Interpolation

A set of control points forms a boundary for a region of space that is called a <u>convex hull</u>





Spline Specifications.

It has 3 different forms.

- Algebraic form (12 algebraic co-efficients)
- Geometric form (end points & tangent vectors)
- 4 point form (four points)

① Algebraic form. – Parametric cubic polynomial representation

$$\left.\begin{aligned} x &= a_{3x} u^3 + a_{2x} u^2 + a_{1x} u + a_{0x} \\ y &= a_{3y} u^3 + a_{2y} u^2 + a_{1y} u + a_{0y} \\ z &= a_{3z} u^3 + a_{2z} u^2 + a_{1z} u + a_{0z} \end{aligned}\right\} \text{12 unknowns}$$

where $0 \leq u \leq 1$

② P(u) is a point vector for a point (x, y, z)

Vector form.

$$p(u) = a_3 u^3 + a_2 u^2 + a_1 u + a_0$$

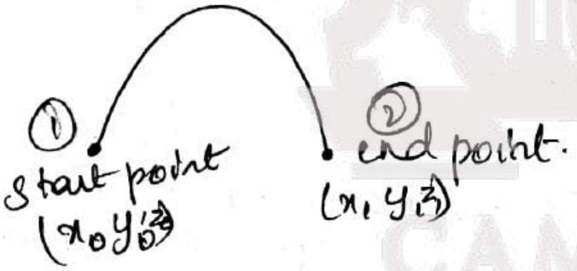
Prof. Supriya S

Boundary for this curve can be set for the endpoint co-ordinate positions $(x_0\ y_0)$ (x_1, y_1) i.e Algebraic to Geometric form

$$x = a_{3x} u^3 + a_{2x} u^2 + a_{1x} u + a_{0x}$$
$$y = a_{3y} u^3 + a_{2y} u^2 + a_{1y} u + a_{0y}$$
$$z = a_{3z} u^3 + a_{2z} u^2 + a_{1z} u + a_{0z}.$$

$$\left.\begin{aligned} x' &= 3a_{3x}u^2 + 2a_{2x}u + a_{1x} \\ y' &= 3a_{3y}u^2 + 2a_{2y}u + a_{1y} \\ z' &= 3a_{3z}u^2 + 2a_{2z}u + a_{1z} \end{aligned}\right\} \text{tangent Vectors.}$$

start & end points with rto tangent vectors.



total 4 points.

Starting tangent vector. ③ $(x_0'\ y_0'\ z_0')$

~~Rto~~ ending tangent vector ④ (x_1', y_1', z_1')

These four boundary conditions are sufficient to determine the values of 4 coefficients a_x, b_x, c_x & d_x

②

To represent in matrix format : $x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$

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

$$= U \cdot C$$

where c is co-efficient matrix for boundary condit^ns

Cubic - Spline Interpolation Methods.

cubic-interpolation splines are obtained by ~~passing~~ connecting passing through every control point as shown in fig

Suppose n+1 control points are specified.

$P_k = (x_k, y_k, z_k)$ $k = 0, 1 \ldots . n.$

with set of equations.

$$\left.\begin{aligned} x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z \end{aligned}\right\} (0 \le u \le 1)$$

* Natural cubic Splines

This interpolation curve is a mathematical representation of original drafting spline.

A natural spline is formulated by requiring the two adjacent curve sections to have same first and second parametric derivatives at their common boundary. Thus natural cubic splines have C^2 continuity.

(0 to n = n+1) → n curve sections

If n+1 control points are considered, then n curve sections with a total of 4n (a, b, c, d) polynomial co-efficients are to be determined.

- The two curve sections on either side of a control point must have same first and second parametric derivations at that control point and each curve must pass through the control point. Thus gives 4n-4 eq^n to be satisfied by 4n polynomial co-efficients.

$P_0 \rightarrow$ first control point (begining of the curve.

$P_n \rightarrow$ last control point

In order to determine values for all the co-efficients

① Method: set second derivatives at P_0 & P_n equal to 0.

② add 2 extra control points (called dummy points), one at the begining of the curve labelled as P_{-1} and the other labelled P_{n+1} at the end.

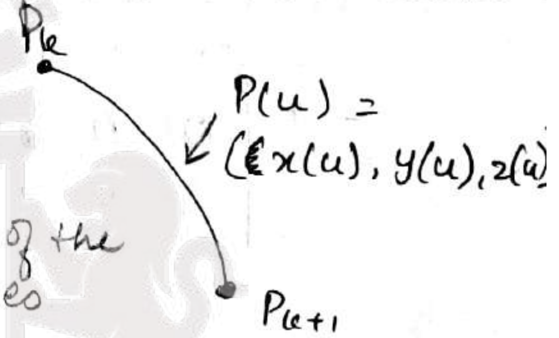
Thus all the original control points are interior & has 4n boundary conditions.

Prof - Supriya S

Hermite Interpolation

Hermite spline named by French Mathematician Charles Hermite) is an interpolating piecewise cubic polynomial with a specified tangent at each control point.

If P(u) represents parametric cubic point-function for curve section b/w point P_k & P_{k+1}, then the boundary conditions of hermite curve section are.



$$P(0) = P_k$$
$$P(1) = P_{k+1}$$
$$\left.\begin{array}{l} P'(0) = DP_k \\ P'(1) = DP_{k+1} \end{array}\right\} \text{Slope of the curves}$$

$\therefore$ $P(u) = au^3 + bu^2 + cu + d \quad 0 \le u \le 1$

equivalent matrix is for $x(u) = a_x u^3 + b_x u^2 + c_x u + d$

$$P(u) = [u^3 \quad u^2 \quad u \quad 1] \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

similarly y & z component

& corresponding tangent vectors.

$$P'(u) = [3u^2 \quad 2u \quad 1 \quad 0] . \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

Substitute $u = 0$ & $u = 1$

$$\begin{bmatrix} P(0) \rightarrow P_k \\ P(1) \rightarrow P_{k+1} \\ P'(0) \rightarrow P'_k \\ P'(1) \rightarrow P'_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} . \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

solving for polynomial co-efficients -

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} \overset{+}{0} & \overset{-}{0} & \overset{+}{0} & \overset{-}{1} \\ \overset{-}{1} & \overset{+}{1} & \overset{-}{1} & \overset{+}{1} \\ \overset{+}{0} & \overset{-}{0} & \overset{+}{1} & \overset{-}{0} \\ \overset{-}{3} & \overset{+}{2} & \overset{-}{1} & \overset{+}{0} \end{bmatrix}^{-1} \cdot \begin{bmatrix} P_k \\ P_{k+1} \\ P'_k \\ P'_{k+1} \end{bmatrix}$$

$$\frac{\text{Adj } A}{|d|}$$

Explanation : Since the first row has 3 column values 0, will only consider the 4th value - 1

$$-1 \begin{vmatrix} \overset{-}{1} & \overset{+}{1} & \overset{-}{1} \\ \overset{+}{0} & \overset{-}{0} & \overset{+}{1} \\ \overset{-}{3} & \overset{+}{2} & \overset{-}{1} \end{vmatrix} = -1 \left[+1 \begin{vmatrix} \overset{-}{1} & \overset{+}{1} \\ \overset{-}{3} & \overset{+}{2} \end{vmatrix} \right]$$

$$= -1 \left(+ \left(-2 + 3 \right) \right)$$

$$= -1 \left(+ \left(+1 \right) \right)$$

$$= -1 \rightarrow \text{determinant}$$

$$\begin{vmatrix} + \begin{vmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 2 & 1 & 0 \end{vmatrix} & - \begin{vmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 3 & 1 & 0 \end{vmatrix} & + \begin{vmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 3 & 2 & 0 \end{vmatrix} & - \begin{vmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 3 & 2 & 1 \end{vmatrix} \\ - \begin{vmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 2 & 1 & 0 \end{vmatrix} & + \begin{vmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 3 & 1 & 0 \end{vmatrix} & - \begin{vmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 3 & 2 & 0 \end{vmatrix} & + \begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 3 & 2 & 1 \end{vmatrix} \\ + \begin{vmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 2 & 1 & 0 \end{vmatrix} & - \begin{vmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 3 & 1 & 0 \end{vmatrix} & + \begin{vmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 3 & 2 & 0 \end{vmatrix} & - \begin{vmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 3 & 2 & 1 \end{vmatrix} \\ - \begin{vmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{vmatrix} & + \begin{vmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{vmatrix} & - \begin{vmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{vmatrix} & + \begin{vmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{vmatrix} \end{vmatrix}^{T}$$

$$= \begin{bmatrix} +(0-2) & -(-(0+3)) & +(0+0) & -(-2+3) \\ -(0-2) & +(-(0+3)) & -(0-0) & +(0) \\ +(-(-1+2)) & -(1-3) & +(-(-2+3)) & -(0) \\ -(-(1+0)) & +(-(-1+0)) & -(0) & +(0) \end{bmatrix}^T$$

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

divide by determinant
-1

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

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

i.e
$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & -1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_k \\ P_{k+1} \\ P'_k \\ P'_{k+1} \end{bmatrix}$$

$$= M_H \cdot \begin{bmatrix} P_k \\ P_{k+1} \\ P'_k \\ P'_{k+1} \end{bmatrix}$$

$$\therefore\ P(u) = [u^3\ u^2\ u\ 1] .\ M_H .\begin{bmatrix} P_k \\ P_{k+1} \\ P'_k \\ P'_{k+1} \end{bmatrix}$$

when solving would produce.

$$P(u) = P_k(2u^3-3u^2+1) + P_{k+1}(-2u^3+3u^2) + \overset{P_1^{u}}{DP_k}(u^3-2u^2+u) + \overset{(P'_{k+1})}{DP_{k+1}}(u^3-u^2)$$

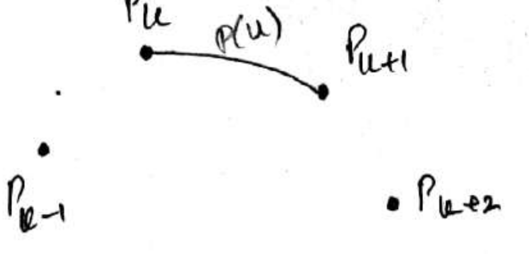
$$= P_k H_0(u) + \underline{P_{k+1} H_1 + DP_k} H_2 + DP_{k+1} H_3$$

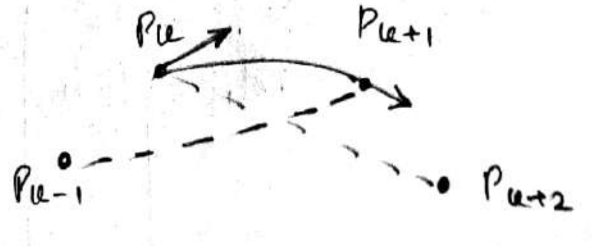
Cardinal Splines

The difference is that we do not input the values for the endpoint tangents.

For a cardinal ~~points~~ spline, the slope at a control point is calculated from the co-ordinates of two adjacent control points.

The figure shows a cardinal spline section with 4 consecutive control point positions. The middle 2 control points are the section endpoints, & other 2 points are used in the calculation of endpoint slopes.





The 4 control points P_{k-1} to P_{k+2} are used to set the boundary conditions for cardinal spline section

$$P(0) = P_k$$
$$P(1) = P_{k+1}$$
$$P'(0) = \frac{1}{2}(1-t)(P_{k+1} - P_{k-1})$$
$$P'(1) = \frac{1}{2}(1-t)(P_{k+2} - P_k)$$

Parameter t is called tension parameter (controls how loosely or tightly the spline fits the points).

when $t=0$, the class of curves is referred to as Catmull-Rom splines or Overhauser splines.

$$P(u) = [u^3 \; u^2 \; u \; 1] . M_c . \begin{bmatrix} P_{k-1} \\ P_k \\ P_{k+1} \\ P_{k+2} \end{bmatrix}$$

(looser curve)
$t<0$

(tighter curve)
$t>0$

Where cardinal matrix is

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

with $s = (1-t)/2$

Prof. Supriya S

Kochanek - Bartels splines.

Extension of cardinal splines.

- 2 additional parameters are introduced into constraint equations defining Kochanek - Bartels splines - to further provide flexibility in adjusting the shapes of curve sections.

$P(0) = P_k$

$P(1) = P_{k+1}$

$$P'(0)_{in} = \frac{1}{2}(1-t)\left[(1+b)(1-c)(P_k - P_{k-1}) + (1-b)(1+c)(P_{k+1}-P_k)\right]$$

$$P'(1)_{out} = \frac{1}{2}(1-t)\left[(1+b)(1+c)(P_{k+1} - P_k) + (1-b)(1-c)(P_{k+2}-P_{k+1})\right]$$

t → tension parameter

b → bias parameter

c → continuity parameter.

- these splines were designed ~~for~~ to model Animation paths. Ex : motion changes used in cartoon Animation

Prof. Supriya S

Bezier Spline curves

- Bezier curves are approximation curve.
- Proposed in 60's by P. Bezier (Pierre)
- used to define sculptured surfaces of automobile bodies (Renault)
- Bezier curves can be fitted to any number of control points (some packages limit to 4 point)
- The degree of Bezier polynomial is determined by no of control points to be approximated & their relative position.

- consider n+1 control points denoted as $P_k = (x_k, y_k, z_k)$ with k varying from 0 to n.

- These co-ordinate points are blended to produce position vector P(u) describing the path between P_0 to P_n

$$\boxed{P(u) = \sum_{k=0}^{n} P_k \, BEZ_{k,n}(u)} \quad , \quad 0 \le u \le 1$$

The Bezier blending functions $BEZ_{k,n}(u)$ are the Bernstein polynomials

$$\boxed{BEZ_{k,n}(u) = C(n,k)\, u^k (1-u)^{n-k}}$$

where parameter C(n,k) are binomial co-efficients $C(n,k) = \frac{n!}{k!\,(n-k)!}$

To explain the above in detail: 4 control point Bezier

Def^n: $x(u) = (1-u)^3 x_0 + 3(1-u)^2 u\, x_1 + 3(1-u)u^2 x_2 + u^3 x_3$

This can be written as

$$x(u) = {}^3C_0 (1-u)^3 x_0 + {}^3C_1 (1-u)^2 u\, x_1 + {}^3C_2 (1-u) u^2 x_2 + {}^3C_3 u^3 x_3$$

$${}^3C_0 = 1, \quad {}^3C_1 = 3 ; \quad {}^3C_2 = 3, \quad {}^3C_3 = \underline{1}$$

$$\therefore \; x(u) = \sum_{i=0}^{3} \underbrace{\overset{\overset{n}{\to}}{{}^3}C_{i_{\to k}} (1-u)^{3-i} u^i}_{BEZ_{k,n}} x_i$$

which can be represented w.r.t control points 0 to n. – (n+1) control points.

$$BEZ_{k,n}(u) = C(n,K)(1-u)^{n-K} \cdot u^K$$

$$\boxed{\therefore \; P(u) = \sum_{k=0}^{n} BEZ_{k,n}(u) \cdot P_k}$$

individually

$$x(u) = \sum_{k=0}^{n} x_k \, BEZ_{k,n}(u)$$

$$y(u) = \sum_{k=0}^{n} y_k BEZ_{k,n}(u)$$

$$z(u) = \sum_{k=0}^{n} z_k \, BEZ_{k,n}(u)$$

for $n \geq k$, Recursive calculations.

$$C(n,k) = \frac{n-k+1}{k} c(n, k-1)$$

Bezier blending functions satisfy recursive relationship

$$BEZ_{k,n}(u) = (1-u)BEZ_{k,n-1}(u) + u\, BEZ_{k-1,n-1}(u)$$

$$n > K \geq 1$$

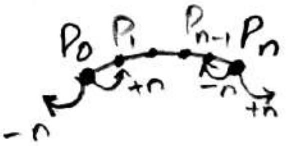
Properties of Bezier curve.

Useful property of Bezier curve is that the curve connects the first and last control points

$$P(0) = P_0$$
$$P(1) = P_n$$

* First ~~degr.~~ derivatives of Bezier curve at the endpoints

$$P'(0) = -nP_0 + nP_1$$
$$P'(1) = -nP_{n-1} + nP_n$$



from the above expressions,
Slope at the begining of the curve is along the line joinig first 2 control points
Slope at the end of the curve is along the line joining last 2 control points/endpoints.

* Second derivatives of Bezier curve - gives curvatures

$$p''(0) = n(n-1)[P_2 - P_1) - (P_1 - P_0)]$$
$$P''(1) = n(n-1)[(P_{n-2} - P_{n-1}) - (P_{n-1} - P_n)]$$

* Important property of Bezier curve is that it lies within the convex hull (convex polygon boundary)

$$\sum_{k=0}^{n} BEZ_{k,n}(u) = 1$$

Cubic bezier curve

curve difinition in matrix form.

$$x(u) = (1-u)^3 x_0 + 3(1-u)^2 u x_1 + 3(1-u) u^2 x_2 + u^3 x_3$$

can be written as.

$$x(u) = (1-3u+3u^2-u^3)x_0 + (3u-6u^2+3u^3)x_1 + (3u^2-3u^3)x_2 + u^3 x_3.$$

$$= (-x_0+3x_1-3x_2+x_3)u^3 + (3x_0-6x_1+3x_2)u^2 + (-3x_0+3x_1)u + x_0$$

$$x(u) = [u^3 \ u^2 \ u \ 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$M_{Bez} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$(a+b)^3 = a^3 + b^3 + 3ab(a+b)$

$(a+b)^2 = a^2 + b^2 + 2ab$

$(a-b)^2 = a^2 + b^2 - 2ab$

Bezier Surfaces

- 2 sets of orthogonal Bezier curves can be used to design an object surface.

- parametric vector function

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

where $P_{j,k}$ specifies the location of $(m+1)$ by $(n+1)$ control points.

- fig shows control points are connected by dashed lines, solid lines shows curves of constant u and constant v.

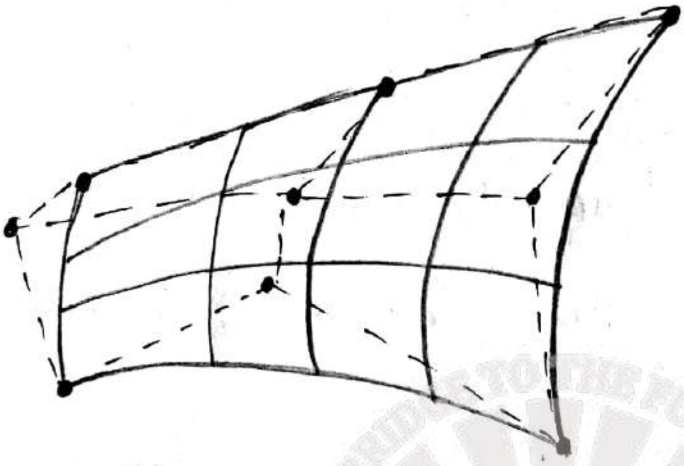
- Each curve of constant u is plotted by varying v over the interval from 0 to 1, with u fixed at one of the values in this unit interval.

- curves of constant v are plotted similarly.

- 3D co-ordinate position for control point can be specified
  - construct a rectangular grid in 'xy' plane / xy "ground" plane.
  - choose elevations above the ground plane at grid intersections as z-co-ordinate value for control point.

- figure illustrates ~~smooth~~ a surface formed with two Bezier sections.
  smooth transition from one section to other is assured by establishing both zero order and first order continuity at boundary line.
  - zero order continuity → matching control points at boundary.

first order continuity → control points along a straight line across boundary.



- - - - control points are connected

——— curve of constant u & v.