Computer Graphics

Contributed By:
Jasaswi Prasad Mohanty

LECTURENOTES.IN

Computer Graphics

Topic:

Overview Of Graphics System

Contributed By:

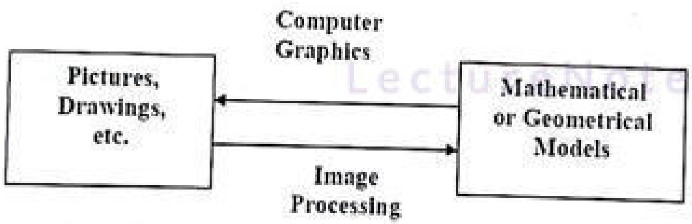
Jasaswi Prasad Mohanty

OVERVIEW OF GRAPHICS SYSTEM

1. INTRODUCTION TO COMPUTER GRAPHICS?

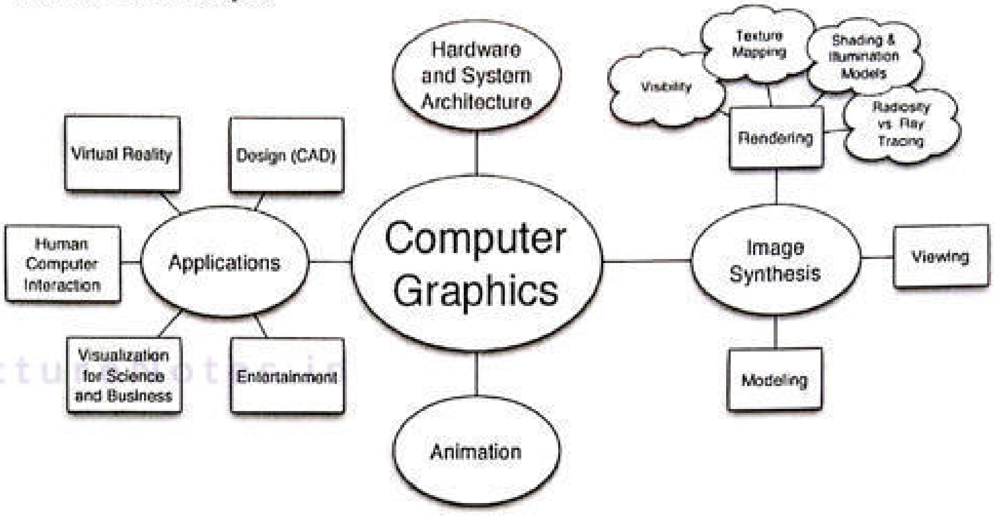
- It is the use of computer to define, store, interrogate and present pictorial output.
- Using a computer as a rendering tool for the generation (from models) and manipulation of images is called computer graphics. More precisely: image synthesis.
- Computer Graphics involves display, manipulation and storage of picture and experimental data for proper visualization using computer.
- **Difference between CG, Image Processing and Computer Vision:**
  - ✓ **Computer Graphics:**
    It is about drawing things on the screen with pixels, using mathematics and physics (trigonometry, lighting, shading, curvature, etc) to give the impression of objects to a human viewer.
    The output requirements can be simple (eg games), or complex (eg realistic rendition for movies.)
  - ✓ **Image Processing :**
    Image Processing is about taking a digital input (black and white photo or colour photo or scanned image or xerox copy etc) and using mathematics and physics (trigonometry, lighting, shading, curvature, etc) to extract details of objects in that input.
    The output requirements can be simple eg finding lines or detecting colours (which can be for non-AI purposes) or complex eg finding faces or detecting emotions (which can be for AI purposes).
  - ✓ **Computer Vision :**
  - ✓ Computer Vision is concerned with the interpretation of video and images. Computer vision takes an incoming image (or series of images) and uses it to create new, non-image information.
    Key examples are:
    - ❑ Creating a 3D model from a video.
    - ❑ Identifying lines and structures from building photos.
    - ❑ Locating and identifying faces (or license plates, or street signs, or…) in random imagery.

  Computer Vision is generally used to analyse and pull semantic content from video and images of the real world.



- **Goal of CG:**
  - ✓ Generate synthetic images (image created by software)
  - ✓ Do it in a practical way and scientifically sound.
  - ✓ In real time?

✓ And make it look easy…

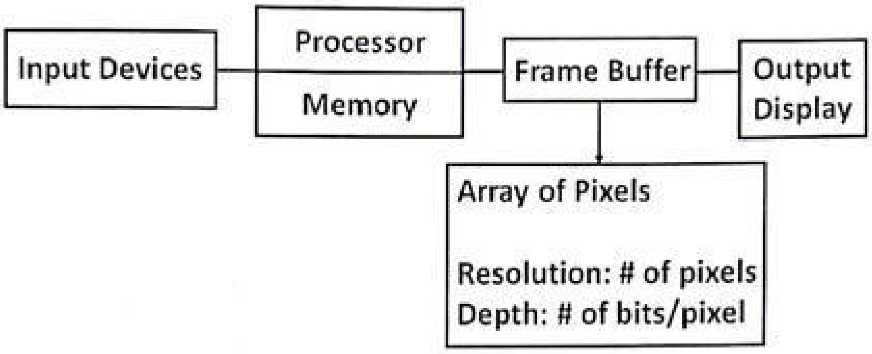


- **Basic Elements:**
  - ✓ **Modelling:**
    - ❑ We try to define the shape (geometry) of the object.
    - ❑ How to represent real environments
      - ▪ Geometry: curves, surfaces, volumes
      - ▪ Photometry: light, colour, reflectance
    - ❑ How to build these representations
      - ▪ Interactive: sculpt it
      - ▪ Algorithmic: let it grow (fractals, extraction)
      - ▪ Scanning: via 3D sensing
      - ▪ Generate primitives – Lines, triangles, quads, patches, Cylinder, spheres
  - ✓ **Rendering** (as realistic as possible)
    - ❑ Way to display (shading, illumination, color, texture …) objects
    - ❑ What is an image? – Distribution of light energy on 2D "film"
      - ▪ How do we represent and store images?
        Sampled array of "pixels": p[x,y]
      - ▪ How do we generate images from scenes?
        Input: 3D description of scene, camera – Project to camera's viewpoint – Illumination
  - ✓ **Animation**
    - ❑ Addresses the issues of movement (dynamics)
    - ❑ Model how things move
    - ❑ Temporal change of
      - ▪ Objects (position, orientation, size, shape, color, etc.)
      - ▪ Camera (position, direction, angle, focus, etc.)
      - ▪ Illumination (position, direction, color, brightness)
- **Applications:**
  - ✓ **Engineering:** We can do simulations using virtual parts (images)
  - ✓ **Medical:** We can use CG as building tools which allow us the visualization of various parts or organs in a human body. So there is a project called visible human project where you have enormous amount of data in digital form, in

slices and you can use that to do 3D reconstruction of parts. We can perform the biomedical simulations; as what happens if I move this skeleton, how this muscle deforms, how does the skin change and so on. So we use CG in the domain of medicine. We have gross level of body structure and minute structures for instance tooth and we can look at the reconstruction of those. Hence they can help us in the process of diagnosis of any abnormality, visualize them so this can be good aid to the medical.

- ✓ **Bio-graphics:** We are dealing with molecules and molecular structures so we can have representations of various parts or atoms of a molecule and their connections or other functional aspects. For instance, if protein is represented, what are the linkages of protein to the rest of the structure and so on? Hence one can also apply graphics in biology.
- ✓ **Entertainment:** Computer Games and movies
- ✓ **Visualization (science, business, etc.):** It is the process of representing data graphically and interacting with these representations in order to gain insight into the data. CG has provided a powerful mechanism for creating, manipulating and interacting with these representations.
- ✓ **Design:** Computer Aided Design is the use of Computer Systems to aid in the creation, modification, analysis or optimization of a design.
- ✓ **Computer Simulation:** Computer simulation reproduce the behaviour of a system using a mathematical model. Simulations have become a useful tool for the mathematical modelling of many natural systems in physics, chemistry, biology, human systems in economics, psychology, social science and engineering.
- ✓ **Web design**
- ✓ **Digital art**
- ✓ **Animation**
- ✓ **Presentation and Training**

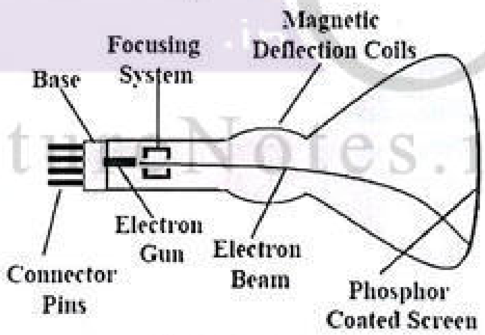
- **Graphics System:**
  - ✓ A Graphics System consists of:
    - ❑ A host computer
    - ❑ A fast processor
    - ❑ Large memory
    - ❑ Frame buffer
    - ❑ Set of input devices (keyboard, mouse, scanner, touchscreen, joystick, trackball)
    - ❑ Output Devices (Printer, Monitor, Plotter)
    - ❑ Set of interface devices (video input/output, TV interface)



- **Various application packages in graphics:**
  - ✓ Core graphics
  - ✓ GKS (Graphics Kernel System)
  - ✓ SRGP (Simple Raster Graphics package)
  - ✓ Open GL (Graphics Library)
- **Set of Computer Graphics Devices:**
  - ✓ CRT (Cathode Ray Tube) Monitor
  - ✓ EGA (Extended Graphics Adaptor)
  - ✓ CGA (Colour Graphics Adaptor)
  - ✓ VGA (Vector Graphics Adaptor)

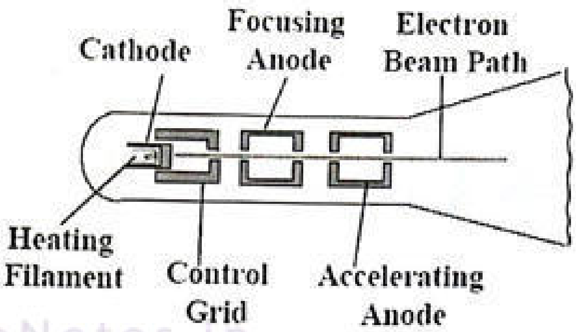
2. VIDEO DISPLAY DEVICES:

- 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) design.
- The following figure shows the basic operation of a CRT:



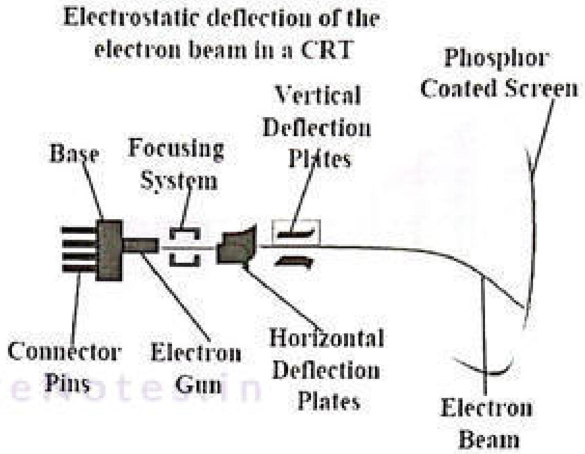
Basic design of a magnetic deflection CRT

- A beam of electrons (cathode rays), emitted by an electron gun, passes through focusing and deflection systems that direct the beam toward specified positions on the phosphor-coated screen.
- **Refresh CRT:** Light emitted by the Phosphor fades very rapidly. To keep the phosphor glowing the picture is redrawn repeatedly by quickly directing the electron beam back over the same point. This type of display is called a refresh CRT.

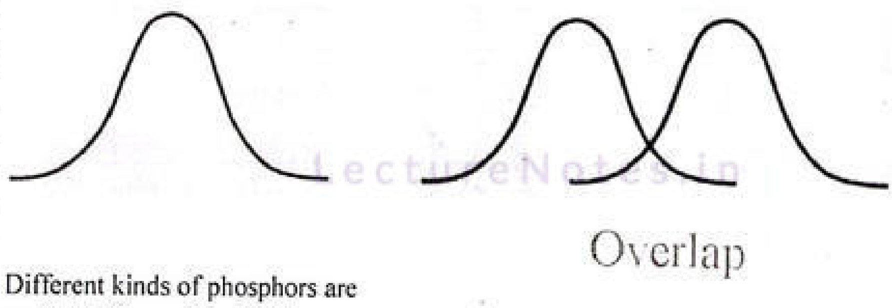


Operation of an electron gun with an accelerating anode

- **Electron gun:** The primary components of an electron gun in a CRT are the **heated metal cathode** and a **control grid.**
  - ✓ **Filament:** Heat is supplied to the cathode tube by directing a current through a coil of wire, called the filament.
  - ✓ This causes the free negatively charged electrons accelerated toward the phosphor coating by a **high positive voltage.**
  - ✓ The accelerating voltage can be generated with a **positively charged metal coating** (shown in wave line in the above figure) on the inside of the CRT envelope near the phosphor screen.
  - ✓ **Intensity** of the electron beam is controlled by setting voltage level on the control grid.
  - ✓ **A smaller negative voltage** on the control grid simply decrease the number of electrons passing through.
- **Focusing System:** The **focusing system** is needed to force the electron beam to converge into a small spot as it strikes the phosphor.
  - ✓ **Electrostatic focusing** is commonly used in television and computer graphics monitor.
  - ✓ With electrostatic focusing, the electron beam passes through a positively charged metal cylinder that forms an **electrostatic lens.**
  - ✓ Similar lens focusing effects can be accomplished with a **magnetic field** set up by a coil mounted around the outside of the CRT envelope.
  - ✓ The **distance** that the electron beam must travel to different points on the screen **varies** because the **radius of curvature** for most CRTs is ***greater*** than the distance from the focusing system to the screen center.
  - ✓ The electron beam will be focused properly only at the **center** of the screen.
  - ✓ As the beam moves to the **outer edges** of the screen, displayed images become ***blurred.***
  - ✓ **Dynamically focusing lens** work based on beam position.



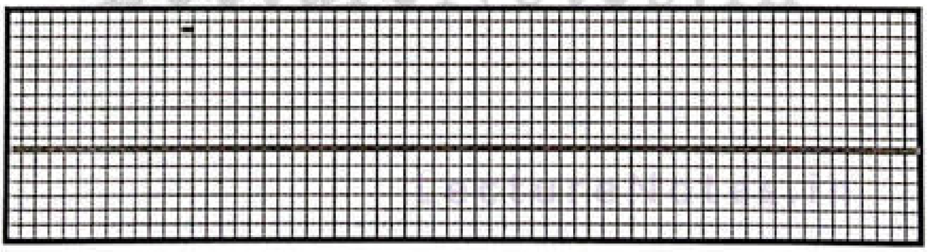
- **Deflection Systems:** Deflection of the electron beam can be controlled either with electric fields or with magnetic fields.
  - ✓ The magnetic deflection coils mounted on the outside of the CRT envelope.
  - ✓ **Two pairs of coils** are used, with the coils in each pair mounted on opposite sides of the neck of the CRT envelope.
    - ❑ One pair is mounted on the top and bottom of the neck, and the other pair is mounted on opposite sides of the neck.
  - ✓ **Horizontal deflection** is accomplished with one pair of coils, and **vertical deflection** by the other pairs.
  - ✓ The proper deflection amounts are attained by adjusting the **current** through the coil.
  - ✓ Two pairs of parallel plates are mounted inside the CRT envelope.
  - ✓ One pair of plates is mounted horizontally to control the **vertical deflection**, and the other pair is mounted vertically to control **horizontal deflection.**
- **Spots of lights** are produced on the screen by the transfer of the CRT beam energy to the phosphor.
  - ✓ Part of the beam energy is converted into heat energy.
  - ✓ The **excited** phosphor electrons begin dropping back to their stable ground state, giving up their extra energy as small quantum of **light energy**.
  - ✓
  - ✓
  - ✓
  - ✓
  - ✓
  - ✓
  - ✓
  - ✓

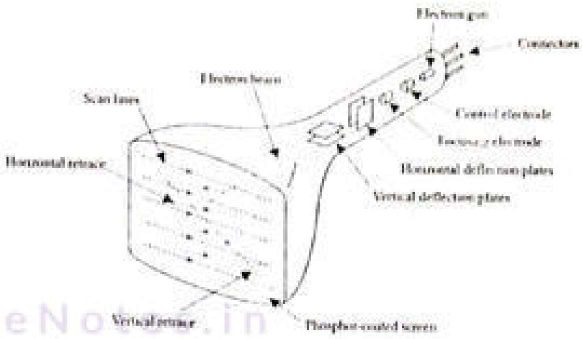


  - ✓ Different kinds of phosphors are available for use in a CRT.
  - ✓ Beside colour, a major difference between phosphors is their persistence (how long they continue to emit light).
  - ✓ **Persistence**: The time it takes the emitted light from the screen to decay to one-tenth of its original intensity.

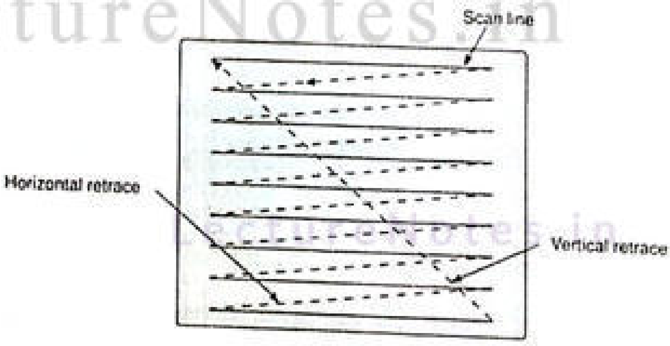
  - ✓ Lower-persistence phosphors require higher refresh rates to maintain a picture on the screen without flicker.
- **Intensity Distribution:** The intensity is greatest at the center of the spot, and decrease with Gaussian distribution out to the edges of the spot.

- **Resolution:** The maximum number of points that can be displayed without overlap on a CRT. It is also defined as the number of points per centimeter that can be plotted horizontally and vertically.
  - ✓ Resolution of a CRT is dependent on:
    - ❑ The type of phosphor
    - ❑ The intensity to be displayed
    - ❑ The focusing and deflection systems.
  - ✓ Typical resolution on high-quality systems is 1280 by 1024.
  - ✓ High resolution systems are often referred to as high-definition systems.
- **Aspect Ratio:** This numbers gives the ratio of vertical points to horizontal points necessary to produce equal length lines in both directions on the screen.
  - ✓ An aspect ratio of 3/4 means that a vertical line plotted with three points has the same length as a horizontal line plotted with four points.
- **Scanning Systems in Graphics:**
  - ✓ Random Scan Display
  - ✓ Raster Scan Display
- **Raster Scan Display:**
  - ✓ **Raster:** A rectangular array of points or dots
  - ✓ **Pixel:** One dot or picture element of the raster
  - ✓ **Scan Line:** A row of pixels





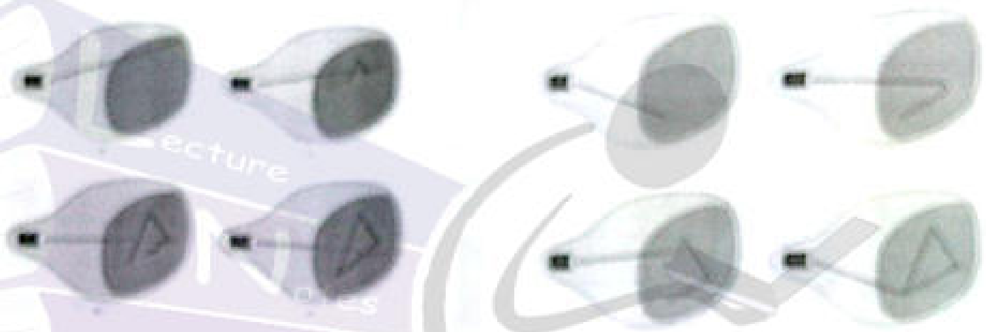
- ✓ In a raster scan system, the electron beam is swept across the screen, one row at a time from top to bottom.
- ✓ As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.
- ✓ Picture definition is stored in a memory area called the **refresh buffer** or **frame buffer**.
- ✓ **Refresh buffer or frame buffer:** This memory area holds the set of intensity values for all the screen points.
- ✓ Intensity range for pixel positions depend on the capability of the raster system.
- ✓ In a black-and-white system: each screen point is either on or off, so only **one bit** per pixel is needed to control the intensity of screen positions.
- ✓ On a black-and-white system with one bit per pixel, the frame buffer is called **bitmap**.
- ✓ For system with multiple bits per pixel, the frame buffer is called **pixmap**.



- ✓ **Horizontal retrace:** The return to the left of the screen, after refreshing each scan line.
- ✓ **Vertical retrace:** At the end of each frame the electron beam returns to the top left corner of the screen to begin the next frame.

- **Raster image**: The **quality** of a raster image is determined by the total number of pixels (**resolution**), and the amount of information in each pixel (**color depth**).
- Raster graphics cannot be **scaled** to a higher resolution without loss of apparent quality.

- **Random Scan Display**
  - ✓ It is known as vector scan.
  - ✓ When operated as a random scan display unit, a CRT has the electron beam directed only to the parts of the screen where a picture is to be drawn.
  - ✓ Random scan monitors draw a picture one line at a time and for this reason are also referred to as **vector displays** (or **Stroke - writing** or **calligraphic** displays).
  - ✓ Random scan display is the use of geometrical primitives such as points, lines, curves, and polygons, which are all based upon **mathematical equation**.
  - ✓ Raster Scan is the representation of images as a collection of **pixels** (dots).



  - ✓ The component lines of a picture can be drawn and refreshed.
  - ✓ **Refresh rate** depends on the number of lines to be displayed.
  - ✓ Picture definition is now stored as a line-drawing commands an area of memory referred to as **refresh display file** (or **display list** or **display program** or **refresh buffer**).
  - ✓ To display a picture, the system cycle through the **set of commands** in the display file, drawing each component line in turn.
  - ✓ Random scan displays are designed to draw all the component lines of a picture 30 to 60 times each second.
  - ✓ Random scan displays are designed for **line-drawing applications** and cannot display realistic shaded scenes.
  - ✓ Random scan displays have higher resolution than raster systems.
  - ✓ Vector displays product smooth line drawing.
  - ✓ A raster system produces jagged lines that are plotted as discrete point sets.
  - ✓ **Example**:

    Data are describing a circle:

    - ❑ *the radius r*
    - ❑ The location of the center point of the circle

- Stroke line style and color
- Fill style and color

✓ **Advantages:**

- This minimal amount of information translates to a much smaller file size. (file size compared to large raster images)
- On zooming in, it remains smooth
- The parameters of objects are stored and can be later modified (transformation).

• **Color CRT Monitors:**

✓ A CRT monitor displays color pictures by using a combination of phosphors that emit different *color* lights.

✓ By combining the emitted light from the different phosphors, a range of colors can be generated.

✓ Basic Techniques for producing color displays:

- Beam-penetration
- Shadow-mask

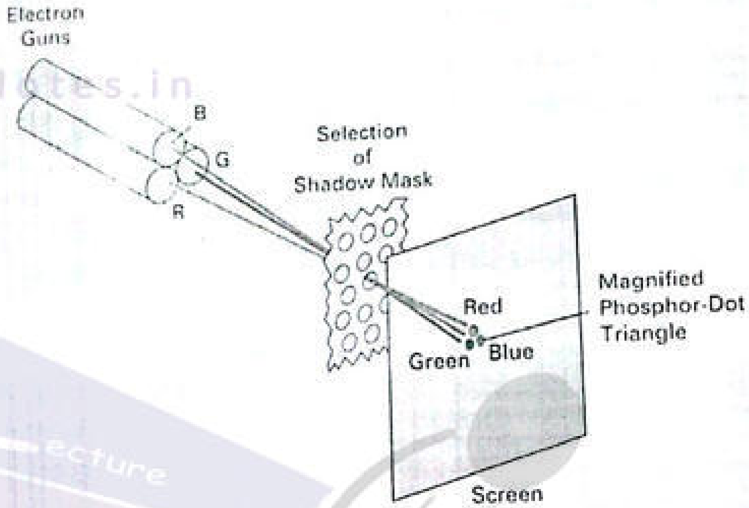
✓ **Beam-penetration Method:**

- This method for displaying colour pictures has been used with random-scan monitors.
- Two layers of ***phosphor*** (**red** and **green**) are coated onto the inside of the CRT screen.
- The displayed color depends on how far the electron beam **penetrates** into the phosphor layers.
- The speed of the electrons, and the screen color at any point, is controlled by the **beam-acceleration voltage.**
- Only four colors are possible (**red**, **green**, **orange**, and **yellow**).
- Quality of pictures is not as good as with other methods.

✓ **Shadow Mask Method**

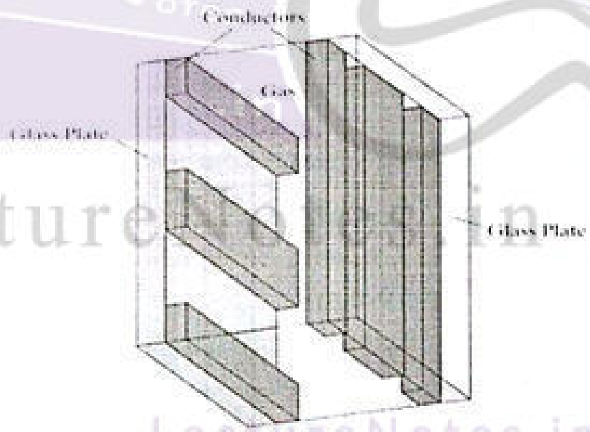
- This method is commonly used in raster-scan systems (including color TV).
- Produce a much wider range of colours than beam-penetration method.
- The color CRT has:
  - Three color ***phosphor*** dots (**red**, **green** and **blue**) at each point on the screen
  - Three ***electron guns***, each controlling the display of red, green and blue light.
- Methods:
  - Delta Method: commonly used in colour CRT systems
  - In-line Method

- ☐ The three electron beams are deflected and focussed as a group onto the shadow mask, which contains a series of holes aligned with the phosphor-dot patterns.
- ☐ We obtain color variations by varying the intensity levels of the three electron beam.
- ☐ Designed as ***RGB*** monitors.
- ☐ High quality raster graphics system have **24** bits per pixel in the frame buffer (a **full color** system or a **true color** system).



- **Direct-View Storage Tubes (DVST):**
  - ✓ Instead of refreshing the screen, DVST stores the picture information inside the CRT just behind the phosphor-coated screen as a charge distribution.
  - ✓ It uses two electron guns: the primary gun stores the picture pattern, the flood gun maintains the picture display.
  - ✓ Advantages:
    - ☐ Because no refreshing is needed, very complex pictures can be displayed at very high resolutions.
  - ✓ Disadvantages:
    - ☐ Ordinarily do not display colors.
    - ☐ Selected parts of a picture cannot be erased. To eliminate a picture section, the entire screen must be erased and the modified picture redrawn.
- **Flat-Panel Displays:**
  - ✓ It is a class of video devices that have reduced volume, less weight and power requirements compared to a CRT.
  - ✓ These are thinner than CRT hence we can hang them on walls or wear them on our wrists.

- ✓ **Example:** small TV monitors, calculators, pocket video games, laptop computers, portable monitors etc.
- ✓ Flat-panel displays can be divided into two categories:
  - ❑ **Emissive displays (emitter):**
    - These devices can converts electrical energy into light.
    - Example: plasma panels, thin-film electroluminescent displays, light-emitting diodes (LED).
  - ❑ **Non-emissive displays (non-emitter):**
    - These devices use optical effects to convert sunlight or light from some other source into graphics patterns.
    - Example: liquid-crystal device (LCD).
- ✓ **Plasma panels** (gas-discharge displays):
  - ❑ These are constructed by filling the region between two gas plates with a mixture of gases that usually includes neon (inert gas).
  - ❑ A series of vertical conducting ribbons is placed on one glass panel and set of horizontal ribbons is built into the other glass panel.
  - ❑ Firing voltages applied to a pair of horizontal and vertical conductors cause the gas at the intersection of the two conductors to break down into a glowing plasma of electrons and ions.
  - ❑ Picture definition is stored in a refresh buffer, and firing voltages are applied to refresh the pixel positions.
  - ❑ **Disadvantage**: Strictly monochromatic

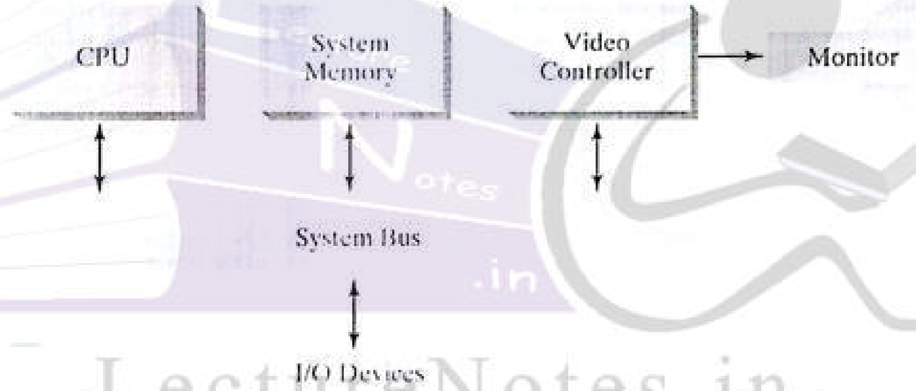


- ✓ **LED:**
  - ❑ A matrix of diodes is arranged to form the pixel positions in the display and picture definition is stored in a refresh-buffer.
  - ❑ Information is read from the refresh buffer and converted to voltage levels that are applied to the diodes to produce the light patterns in the display.
- ✓ **LCD:**

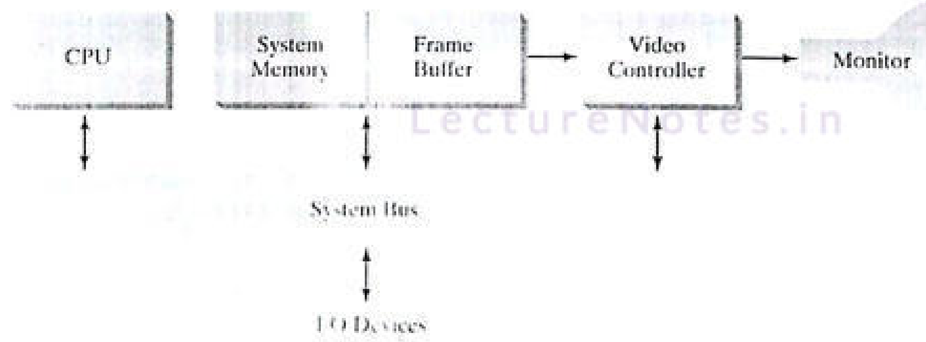
- These devices produce the picture by passing polarized light from the surrounding or from an internal light source through a liquid-crystal material that can be aligned to either block or transmit the light.
- Liquid-crystal refers to the fact that these compounds have a crystalline arrangement of molecules which flow like a liquid.
- Commonly used in small systems such as calculators, laptops etc.
- Types:
    - **Passive-matrix LCD:** The light is reflected back to the viewer. To turn off the pixel, we apply a voltage to the two intersecting conductors to align the molecules so that light is not twisted.
    - **Active-matrix LCD:** Transistors are used at each pixel location which control the voltage at pixel locations to prevent charge from gradually leaking out of the liquid-crystal cells.

3. RASTER SCAN SYSTEMS:

- In addition to the central processing unit (CPU), a special 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 below:

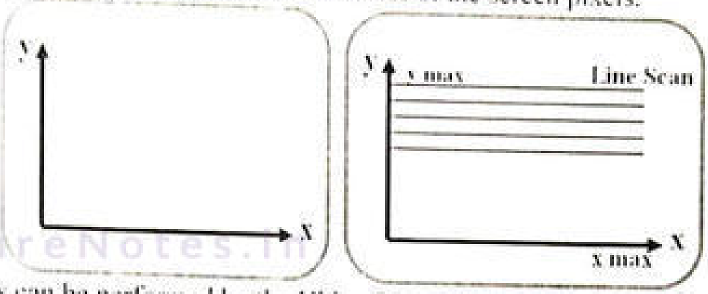


- **Video Controller:** A fixed area of the system memory is reserved for the frame buffer, and the video controller is given direct access to the frame buffer memory.

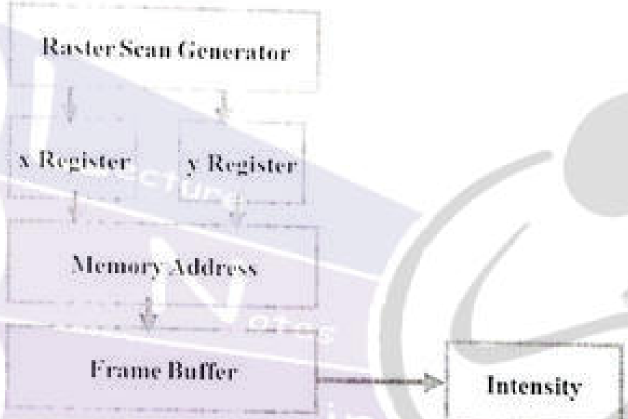


- **Frame buffer** location, and the corresponding screen positions, are referenced in Cartesian coordinates.

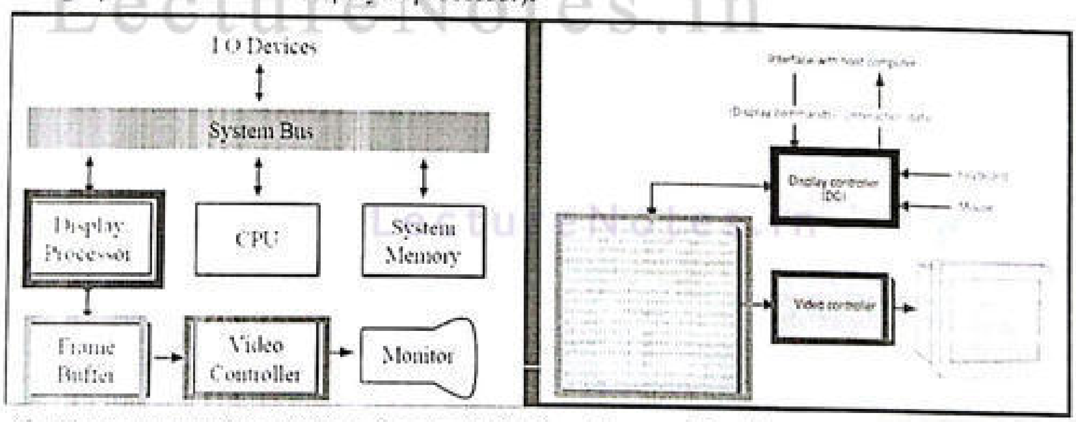
- Scan lines are then labeled from y_{max} at the top of the screen to 0 at the bottom. Along each scan line, screen **pixel** positions are labeled from 0 to x_{max}.
- Two registers are used to store the coordinates of the screen pixels.



- Operations can be performed by the Video Controller:
  - ✓ Refreshing operation
  - ✓ Transformation (Areas of the screen can be enlarged, reduces, or moved during the refresh cycles)



- **Raster Scan Display Processor (DP):**
  - ✓ A raster system containing a separate **display processor** (also referred as graphics controller / display coprocessor).

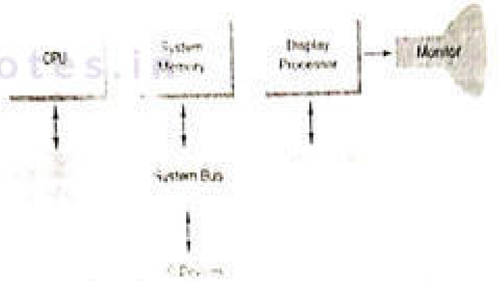


  - ✓ The purpose of the **DP** is to free the CPU from the graphics chores.
  - ✓ A major task of the display processor is **Scan Conversion.**

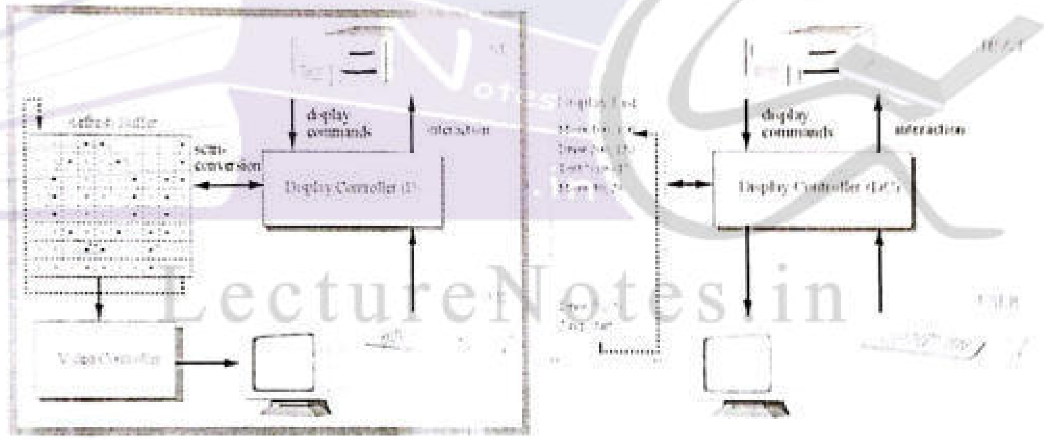
- ✓ **Scan Conversion:** is digitizing a picture definition given in an application program into a set of pixel intensity values for storage in the frame buffer.
- ✓ DP performs the following additional operations:
  - ❑ Generation various line styles (dashed, dotted, or solid)
  - ❑ Displaying color areas
  - ❑ Performing certain transformation and manipulation on display objects.

4 RANDOM SCAN SYSTEMS:

- The organization of a simple random-scan (vector) system is shown below:



- Graphic commands are translated by the graphics package into a display file stored in the system memory.
- This file is then accessed by the **display processor unit (DPU)** (also referred as graphic controller) to refresh the screen.
- Difference between Raster Scan and Random Scan:



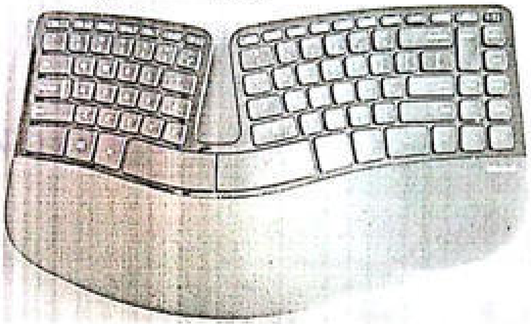
5. INPUT DEVICES:

- These devices are used for data input on graphics workstations.
- Some of these are:

5.1 Keyboard:

- An alphanumeric keyboard in a graphics system is used primarily as a device for entering text strings.
- It is mainly used for entering nongraphic data.
- It facilitate the entry of screen coordinates, menu selections, or graphic functions.

- Cursor-control keys are used to select displayed objects or coordinate positions by positioning the moving cursor.
- Function keys allow users to enter frequently used operations in a single keystroke.
- For specialized applications, input to a graphics application may come from a set of buttons, dials or switches that select data values or customized graphics operations.
- Buttons and switches are often used to input predefined functions, and dials are common devices for entering scalar values.



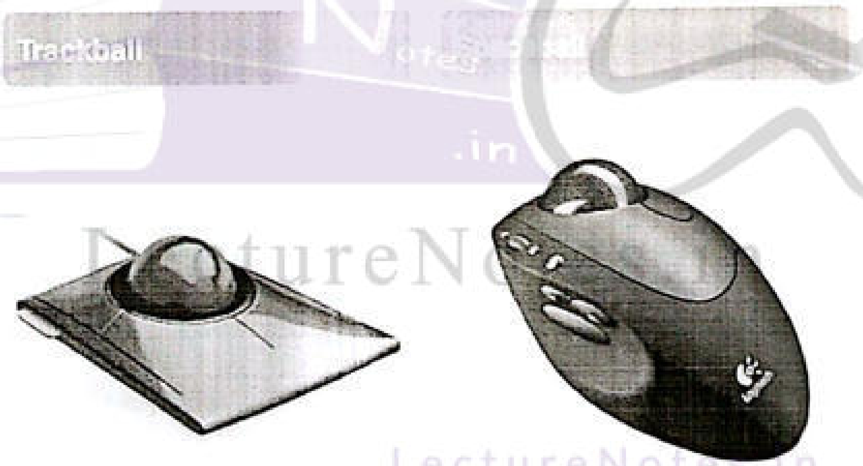
- Real numbers within some defined ranges are selected for input with dial rotations.
- Potentiometers are used to measure dial rotation, which are then converted to deflection voltages for cursor movements.

5.2 Mouse:

- A mouse is a small hand-held box used to position the screen cursor.
- Rollers on the bottom of the mouse can be used to record the amount and direction of movement.
- Mouse motion can be detected with an optical sensor. In this system mouse is moved over a special mouse pad that has a grid of horizontal and vertical lines. The optical sensor detects the movement across the lines in the grid.
- One, two, or three buttons are usually included on top of the mouse for signaling the execution of some operations.
- The Z-mouse is a special type of mouse which includes three buttons, a thumbwheel on the side, a trackball on the top, and a standard mouse ball underneath.
- With a Z-mouse, we can pick up an object, rotate it and move it in any direction, or we can navigate our viewing position and orientation through a three-dimensional scene.
- Application of Z-mouse include virtual reality, CAD and animation.

5.3 Trackball and Spaceball:

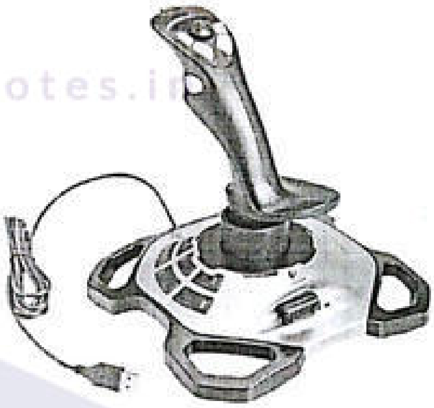
- It is a ball that can be rotated with fingers or palm to produce screen cursor movement.
- Potentiometers, attached to the ball measure the amount and direction of rotation.
- Sometime trackball are attached on keyboards or Z-mouse.
- Spaceballs are used for three-dimensional positioning and selection operations in virtual-reality systems, modelling, animation, CAD etc.
- A spaceball does not actually move.
- Strain gauges measure the amount of pressure applied to the spaceball to provide input for special positioning and orientation as the ball is pushed or pulled in various directions.



5.4 Joysticks:

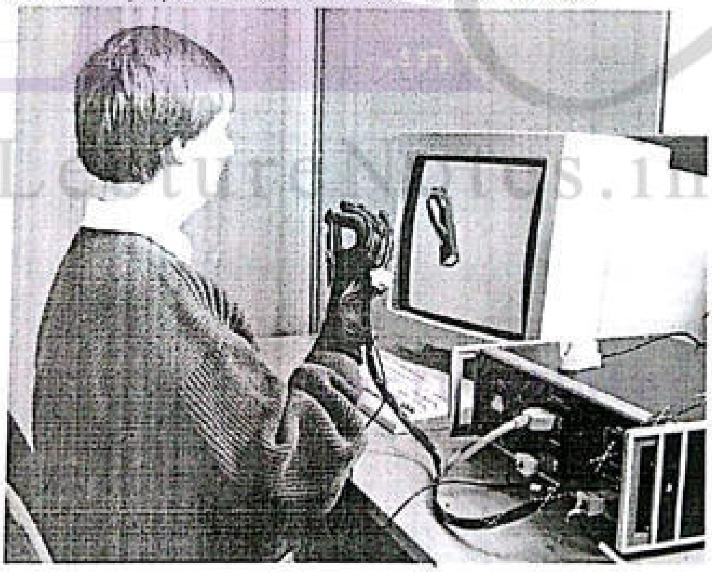
- Joystick consists of a small, vertical lever (stick) mounted on a base that is used to steer the screen cursor around.
- Most joysticks select screen positions with actual stick movement, others respond to pressure on the stick.
- Some joysticks are mounted on the keyboard and some functions as stand-alone units.

- The distance that the stick is moved in any direction from its center position corresponds to screen-cursor movement in that direction.
- Potentiometers mounted at the base of the joystick measure the amount of movement, and springs return the stick to the center position when it is released.
- Pressure-sensitive joysticks have a non-movable stick. Pressure on the stick is measured with strain gauges and converted to movement of the cursor in the direction specified.



5.5 Data Glove:

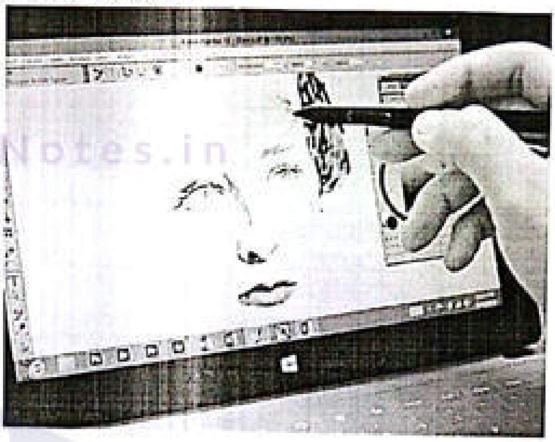
- Data Glove is used to grasp a "virtual" object.
- The globe is constructed with a series of sensors that detect hand and finger motions.
- Inputs from the globe can be used to position or manipulate objects in a virtual scene.
- A two-dimensional projection of the scene can be viewed on a video monitor, or a three-dimensional projection can be viewed with a headset.



5.6 Digitizers:

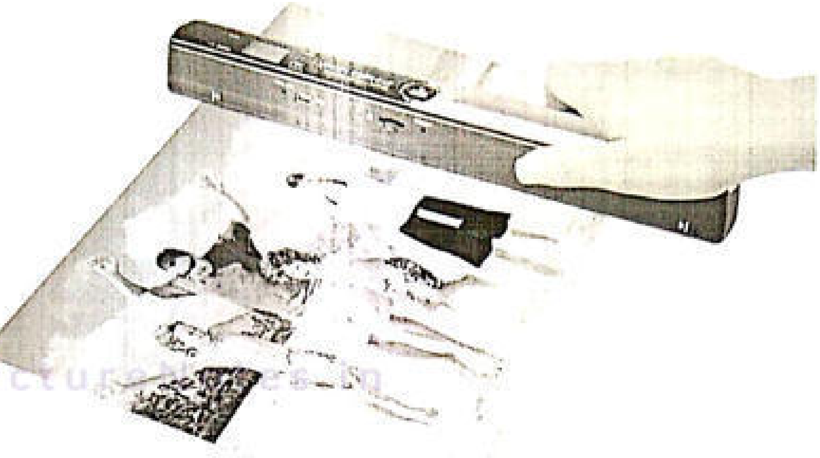
- A digitizer tablet (also known as a digitizer or **graphics** tablet) is a tool used to convert hand-drawn images into a format suitable for **computer** processing.

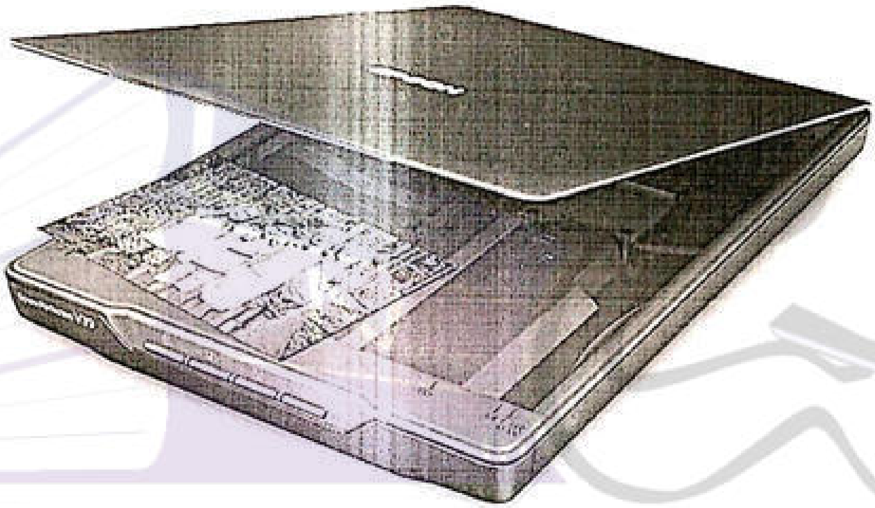
- Images are usually drawn onto a flat surface with a stylus (a small pen-shaped instrument that is used to input commands to a **computer** screen.) and then appear on a **computer** monitor or screen.
- Digitizer tablets can also be used as an input device, receiving information represented in drawings and sending output to a CAD (computer aided design) application and PC-based software like AutoCAD.



5.7 Image Scanners:

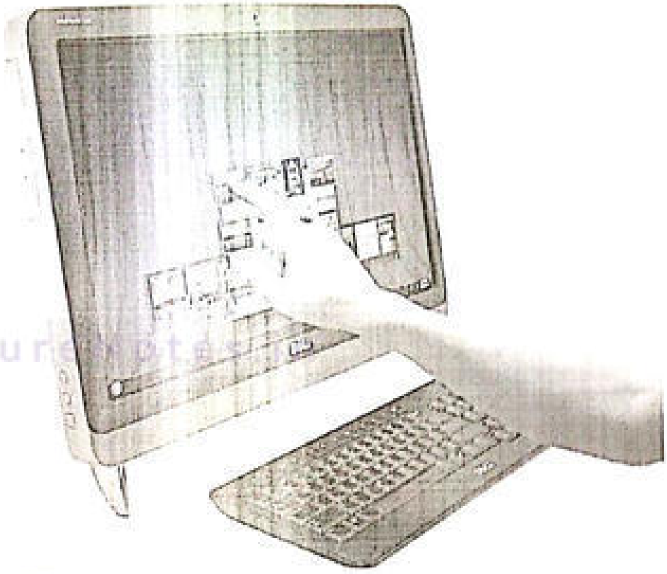
- Drawing, graphs, colour and black-and-white photos, or text can be stored for computer processing with an image scanner by passing an optical scanning mechanism over the information to be stored.
- The gradations of gray scale or colour are recorded and stored in an array.
- Using the information stored in the array the picture can be rotated, scaled or cropped to a particular screen area.
- Some scanners are able to scan either graphical representations or text, and they come in a variety of sizes and capabilities.





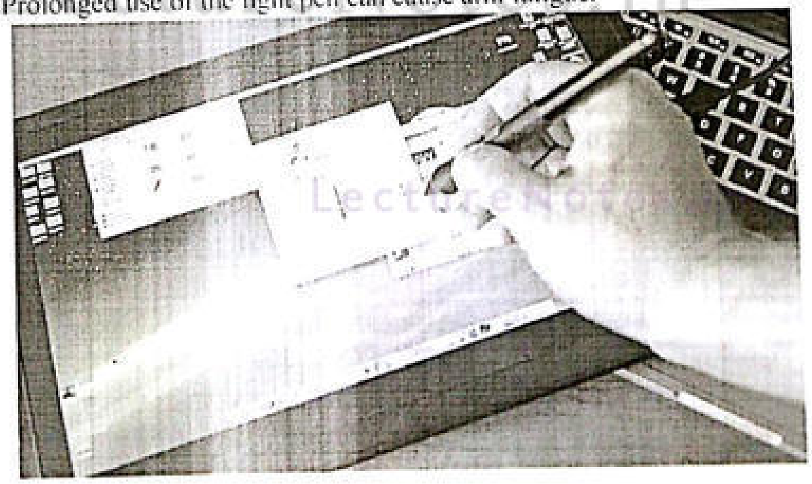
5.8 Touch Panels:

- Touch Panels allow displayed objects or screen positions to be selected with a touch of a finger.
- Normally touch panel is used for the selection of processing options that are represented with graphical icons.
- Touch input can be recorded using optical, electrical or acoustical methods.



5.9 Light Pens:

- Light pens are used to select screen positions by detecting the light coming from points on the CRT screen.
- An activated light pen, pointed at a spot on the screen as the electron beam lights up that spot, generates an electrical pulse that causes the coordinate position of the electron beam to be recorded.
- Recorded light-pen coordinates can be used to position an object or to select a processing option.
- Disadvantages:
  - ✓ Light pens require special implementations for some applications because they cannot detect positions within black areas.
  - ✓ Light pens sometimes give false readings due to background lighting in a room.
  - ✓ Prolonged use of the light pen can cause arm fatigue.



5.10 Voice Systems:

- Speech recognizers are used in some graphics workstations as input devices to accept voice commands.
- The voice-system input can be used to initiate graphics operations or to enter data.
- These systems operate by matching an input against a predefined dictionary of words and phrases.
- A dictionary is set up for a particular operator by having the operator speak the command words to be used into the system several times.
- When a voice command is given, the system searches the dictionary for a frequency-pattern match.
- If a different operator is to use the system, the dictionary must reestablished with that operator's voice patterns.
- Advantage: The attention of the operator does not have to be switched from one device to another to enter a command.

4. HARD-COPY DEVICES:

- We can obtain hard-copy output for our images in the following formats:
  - ✓ For presentation we can produce overhead transparencies (35-mm sides)
  - ✓ To put images on film, we can photograph a scene displayed on a video monitor.
  - ✓ We can put pictures on papers by directing graphics output to printer of plotter.
- The quality of picture depends on dot size and the number of dots per inch, or lines per inch, that can be displayed.
- Printers produce output by using the following methods:
  - ✓ Impact:
    - ❑ Impact printers press formed character faces against an inked ribbon onto the paper.
    - ❑ Example: Line printer with the typefaces mounted on bands, chains, drums, or wheels.
  - ✓ Non-impact:
    - ❑ Nonimpact printers and plotters use laser techniques, ink-jet sprays, xerographic processes, electrostatic methods and electrothermal methods to get images onto paper.
- Character impact printers often have a dot-matrix print head containing a rectangular array of protruding wire pins, with the number of pins depending on the quality of the printer. Individual characters or graphics patterns are obtained by retracting certain pins so that the remaining pins form the pattern to be printed.
- In a laser device, a laser beam creates a charge distribution on a rotating drum coated with a photoelectric material, such as selenium. Toner is applied to the drum and then transferred to paper.
- Ink-jet methods produce output by squirting ink in horizontal rows across a roll of paper wrapped on a drum. The electrically charged ink stream is deflected by an electric field to produce dot-matrix patterns.
- In impact printer we can get limited color output by using different colored ribbons.

- Non-impact devices use various techniques to combine three color pigments (cyan, magenta and yellow) to produce range of color patterns.
- Drafting layouts and other drawings are typically generated with ink-jet or pen plotters.

Computer Graphics

Topic:
Output Primitives

Contributed By:
Jasaswi Prasad Mohanty

Output Primitives

A computer has to take care of 2 things while plotting anything into the screen
- pixels
- computations

Line Drawing Algorithm

The cartesian slope-intercept equation for a straight line is

$$y = mx + c$$

where m is the slope of the line
and c is the y-intercept

Let (x_1, y_1) and (x_2, y_2) be the two end points of a line segment.

We can calculate m and c as follows:

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$$

$$c = y_1 - m.x_1$$

For any given x interval Δx along a line, we can compute the corresponding y interval Δy as follows:

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

Similarly for any given y interval Δy we can compute the corresponding x interval Δx as follows:

$$\Delta x = \frac{\Delta y}{m}$$

Digital Differential Analyzer (DDA) Algorithm

- It is a scan-conversion line algorithm based on calculating either Δx or Δy
- We sample the line at unit intervals in one coordinate and determine corresponding integers values nearest the line path for the other coordinate.
- Let the line is moving from point (x_k, y_k) to (x_{k+1}, y_{k+1})

So $$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} = \frac{\Delta y}{\Delta x}$$

For slope $m < 1$ (x is increasing more than y)

We take $\Delta x = 1$

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

$\Rightarrow \boxed{x_{k+1} = x_k + 1}$

$m = \dfrac{y_{k+1} - y_k}{1} \Rightarrow \boxed{y_{k+1} = y_k + m} \Rightarrow y_{k+1} = y_k + \dfrac{\Delta y}{\Delta x}$

$\Rightarrow \boxed{y_{k+1} = y_k + \dfrac{\Delta y}{step}}$ Here step $= \Delta x$

For slope $m > 1$ (y is increasing more than x)

We take $\Delta y = 1$

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

$\Rightarrow \boxed{y_{k+1} = y_k + 1}$

$m = \dfrac{1}{x_{k+1} - x_k} \Rightarrow x_{k+1} - x_k = \dfrac{1}{m} \Rightarrow x_{k+1} = x_k + \dfrac{1}{m}$

$\Rightarrow x_{k+1} = x_k + \dfrac{\Delta x}{\Delta y} \Rightarrow \boxed{x_{k+1} = x_k + \dfrac{\Delta x}{step}}$

Here step $= \Delta y$

If ($\Delta x > \Delta y$)
  step = abs(Δx)
else
  step = abs(Δy)

DDA Algorithm

Step 1 : Input the coordinate of the two end points $A(x_1, y_1)$ and $B(x_2, y_2)$ for the line AB respectively. Note that points A and B are not equal. If they are equal then it is a point. Plot the point and return.

Step 2 : Calculate dx and dy
$dx = x_2 - x_1$ and $dy = y_2 - y_1$

Step 3: Calculate step

If $abs(dx) \geq abs(dy)$
then $step = abs(dx)$
else $step = abs(dy)$

Step 4: Calculate the increment factor

$$x_{inc} = \frac{dx}{step}$$

$$y_{inc} = \frac{dy}{step}$$

Step 5: Initialize the initial point on the line and plot

$x = x_1$ & $y = y_1$
plot (x, y)

Step 6: Obtain the new pixel on the line and plot

```
for k ← 0 to step-1
    do x ← x + x_inc
       y ← y + y_inc
       Plot( round(x), round(y))
```

Step 7: finish

Rounding

The round(x) function rounds x to the next integer if the value after decimal is greater than or equal to 0.5

```
float round(float a)
{
    int b = a+0.5
    return floor(b)
}
```

Advantage

- simple & fast

Drawback

- Since it uses floating point operation to calculate the pixel/point, so this method is expensive
- It uses rounding function to get the nearest integer.

Problem Using DDA linedrawing algorithm, generate the points between the end points (6, 9) and (11, 19)

Soln
$(x_1, y_1) = (6, 9) \quad (x_2, y_2) = (11, 19)$

$dx = 11 - 6 = 5 \quad , \quad dy = 19 - 9 = 10$

As $abs(dy) = 10 > abs(dx) = 5$

$step = abs(dy) = 10$

$x_{inc} = \frac{5}{10} = 0.5 \quad , \quad y_{inc} = \frac{10}{10} = 1$

$x = 6$, $y = 9$ is the initial point

10 times calculate x & y since $step = 10$

$x = 6 + 0.5 = 6.5 \quad y = y + 1 = 9 + 1 = 10$

$x = 6.5 + 0.5 = 7.0 \quad 11$

x	y
6.5	10
7.0	11
7.5	12
8.0	13
8.5	14
9.0	15
9.5	16
10.0	17
10.5	18
11.0	19

Bresenham's Line Drawing Algorithm

Let us consider a line $y = mx + c$. Assume that $m < 1$

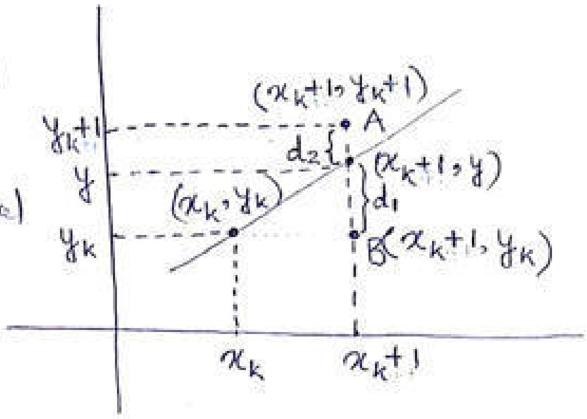
Pixel positions along a line path are determined by sampling at unit x intervals. Starting from the left end point (x_0, y_0) of a given line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path

The figure in the next page demonstrates the k^{th} step

Let us assume that we have determined that pixel (x_k, y_k) is to be displayed, we have to decide which pixel to plot in column $x_k + 1$.

We have two choices $(x_k + 1, y_k)$ and $(x_k + 1, y_k + 1)$

We label vertical pixel separations from the mathematical line path as d_1 and d_2.



The y-coordinate on the mathematical line at pixel column position x_k+1 is calculated as:

$$y = m(x_k+1) + c$$

Then $d_1 = y - y_k = m(x_k+1) + c - y_k$

and $d_2 = (y_k+1) - y = y_k + 1 - m(x_k+1) - c$

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

Substituting $m = \frac{\Delta y}{\Delta x}$

$$d_1 - d_2 = 2\frac{\Delta y}{\Delta x}(x_k+1) - 2y_k + 2c - 1$$

$$\Rightarrow \Delta x(d_1 - d_2) = 2\Delta y\, x_k + 2\Delta y - 2y_k\Delta x + 2c\Delta x - \Delta x$$

Let $P_k = \Delta x(d_1 - d_2)$ where P_k is called the decision variable.

<u>case 1</u>: if $P_k > 0$ then $d_1 > d_2$

Choose the point $A(x_k+1, y_k+1)$

<u>case 2</u>: if $P_k < 0$ then $d_1 < d_2$

Choose the point $B(x_k+1, y_k)$

<u>case 3</u>: if $P_k = 0$ then $d_1 = d_2$

Choose any point either A or B.

Similarly we can find P_{k+1} as

$$P_{k+1} = 2\Delta y\, x_{k+1} + 2\Delta y - 2y_{k+1}\Delta x + 2c\Delta x - \Delta x$$

$$= 2\Delta y(x_k+1) + 2\Delta y - 2y_{k+1}\Delta x + 2c\Delta x - \Delta x$$

$$P_{k+1} - P_k = 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

$$\Rightarrow P_{k+1} = P_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

Note that $y_{k+1} - y_k$ will be either 1 or 0

Case 1: If we choose point $A(x_k+1, y_k+1)$

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

So $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

Case 2: If we choose point $B(x_k+1, y_k)$

then $y_{k+1} - y_k = 0$

So $P_{k+1} = P_k + 2\Delta y$

Case 3: Choose either A or B

for point A, $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

for point B, $P_{k+1} = P_k + 2\Delta y$

Calculation of Initial Decision Parameter P_0

Let us assume that the line $y = mx + c$ passes through (x_0, y_0)

So we have $y_0 = mx_0 + c$

$\Rightarrow c = y_0 - mx_0$

$\Rightarrow c = y_0 - \frac{\Delta y}{\Delta x} x_0$

$\Rightarrow c\Delta x = y_0 \Delta x - \Delta y\, x_0$

$\Rightarrow 2c\Delta x = 2y_0 \Delta x - 2\Delta y\, x_0$

We know already, $P_k = 2\Delta y\, x_k - 2\Delta x\, y_k + 2\Delta y + 2c\Delta x - \Delta x$

Putting $k = 0$, $P_0 = 2\Delta y\, x_0 - 2\Delta x\, y_0 + 2\Delta y + \underline{2c\Delta x} - \Delta x$

$= 2\Delta y x_0 - 2\Delta x y_0 + \underline{2y_0 \Delta x - 2\Delta y x_0} + 2\Delta y - \Delta x$

$= 2\Delta y - \Delta x$

Algorithm (for $m < 1$)

1. Input the two end points of the line and store the left end point in (x_0, y_0)
2. Plot the first point (x_0, y_0)
3. Calculate Δx, Δy, $2\Delta y$, $2\Delta y - 2\Delta x$ and obtain the starting value for the decision parameter as

   $P_0 = 2\Delta y - \Delta x$

4. At each x_k along the line, starting at $k=0$, perform the following tests:

   If $P_k < 0$, the next point to plot is (x_k+1, y_k) and
   $$P_{k+1} = P_k + 2\Delta y$$
   Otherwise, the next point to plot is (x_k+1, y_k+1) and
   $$P_{k+1} = P_k + 2\Delta y - 2\Delta x$$

5. Repeat step 4 Δx times.

Example: Illustrate the Bresenham's Line Drawing Algorithm with end points (20, 10) and (30, 18)

Solⁿ

$$m = \frac{18-10}{30-20} = 0.8 < 1$$

$\Delta x = 30 - 20 = 10$

$\Delta y = 18 - 10 = 8$

$2\Delta y - 2\Delta x = -4$

$2\Delta y = 16$

$P_0 = 2\Delta y - \Delta x = 16 - 10 = 6$

We plot the initial point $(x_0, y_0) = (20, 10)$ and determine the successive pixel positions along the line path 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)

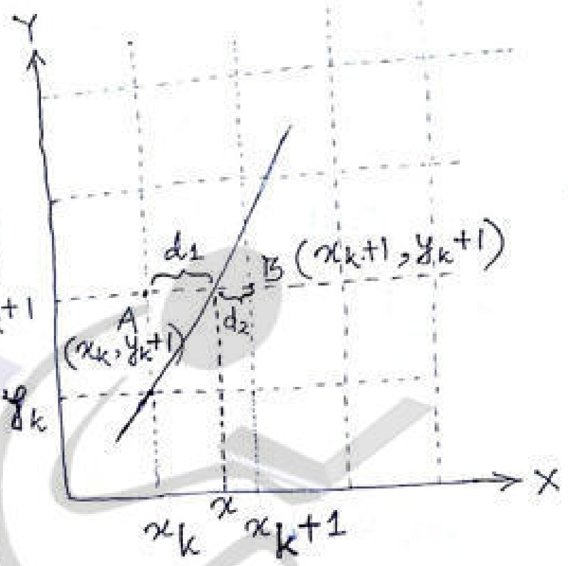
Implementation of Bresenham's Line Drawing Algorithm

```
#include "device.h"
void lineBres(int xa ,int ya ,int xb, int yb)
{
    int dx = abs(xa-xb) , dy = abs(ya-yb);
    int p = 2 * dy - dx ;
    int twoDy = 2*dy , twoDyDx = 2 * (dy - dx)
    int x, y , xEnd ;
    if (xa > xb)   // Determines which point to start with
    {
        x = xb ;
        y = yb ;
        xEnd = xa ;
    }
    else
    {
        x = xa ;
        y = ya ;
        xEnd = xb ;
    }
    setPixel (x, y) ;
    while (x < xEnd)
    {
        x++ ;
        if (p<0)
            p = p + twoDy ;
        else
        {
            y++ ;
            p = p + twoDyDx ;
        }
        setPixel (x,y) ;
    }
}
```

Bresenham's Line Drawing Algorithm for slope, $m > 1$

Here we step along the y direction in unit steps and calculate successive x values nearest the line path. We have determined that pixel (x_k, y_k) is to be displayed, we have to decide which pixel to plot in row y_{k+1}. We have two choices (x_k, y_k+1) and (x_k+1, y_k+1)

The x-coordinate on the mathematical line at pixel row position y_k+1 is calculated as:



$y_k+1 = mx + c$

$\Rightarrow x = (y_k + 1 - c)/m$

$d_1 = x - x_k = (y_k + 1 - c)/m - x_k$

$d_2 = (x_k+1) - x = x_k + 1 - \frac{y_k+1-c}{m}$

$d_1 - d_2 = \frac{y_k+1-c}{m} - x_k - x_k - 1 + \frac{y_k+1-c}{m}$

$= 2\cdot\left[\frac{y_k+1-c}{m}\right] - 2x_k - 1$

$= \frac{2y_k}{m} + \frac{2}{m} - \frac{2c}{m} - 2x_k - 1$

$= 2\frac{y_k}{m} - 2x_k + \frac{2}{m} - \frac{2c}{m} - 1$

Replacing $m = \frac{dy}{dx}$

$d_1 - d_2 = \frac{2y_k dx}{dy} - 2x_k + \frac{2dx}{dy} - \frac{2c\,dx}{dy} - 1$

$\Rightarrow dy(d_1 - d_2) = 2dx\,y_k - 2dy\,x_k + 2dx - 2c\,dx - dy$

Let $P_k = dy(d_1 - d_2)$ where P_k is the decision variable

So we have:

$P_k = 2dx\,y_k - 2dy\,x_k + 2dx - 2c\,dx - dy$

case 1: if $P_k > 0$ then $d_1 > d_2$
Choose the point $B(x_k+1, y_k+1)$

case 2: if $P_k < 0$ then $d_1 < d_2$
Choose the point $A(x_k, y_k+1)$

case 3: if $P_k = 0$ then $d_1 = d_2$
Choose any point either A or B.

We can also find

$$P_{k+1} = 2dx\, y_{k+1} - 2dy\, x_{k+1} + 2dx - 2cdx - dy$$

So $\quad P_{k+1} - P_k = 2dx(y_{k+1} - y_k) - 2dy(x_{k+1} - x_k)$

$$\Rightarrow P_{k+1} = P_k + 2dx - 2dy(x_{k+1} - x_k)$$

Note that the value of $x_{k+1} - x_k$ will be either 1 or 0

case 1 If we choose the point $B(x_k+1, y_k+1)$

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

So $\quad P_{k+1} = P_k + 2dx - 2dy$

case 2 If we choose the point A (x_k, y_k+1)

$$x_{k+1} - x_k = 0$$

So $\quad P_{k+1} = P_k + 2dx$

case 3 Choose either A or B

for point A, $P_{k+1} = P_k + 2dx$

for point B, $P_{k+1} = P_k + 2dx - 2dy$.

Calculation of Initial Decision Parameter P_0

We have $\quad P_k = 2dx\, y_k - 2dy\, x_k + 2dx - 2cdx - dy$

Putting $k=0$, $P_0 = 2dx\, y_0 - 2dy\, x_0 + 2dx - 2cdx - dy$

We know the equation of the line is $y = mx + c$

So we have $\quad y_0 = mx_0 + c$

$$\Rightarrow c = y_0 - mx_0$$

$$= y_0 - \frac{dy}{dx} x_0$$

Putting the value c in the above eqn we have

$$P_0 = 2dx\,y_0 - 2dy\,x_0 + 2dx - 2dx\,y_0 + 2dy\,x_0 - dy$$

$$\Rightarrow P_0 = 2dx - dy$$

Algorithm

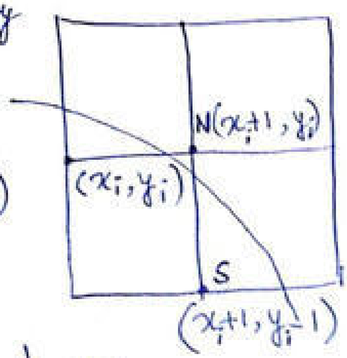
1. Input the two end points of the line and store the left end point as (x_0, y_0)
2. Plot the first end point (x_0, y_0)
3. Calculate dx, dy, 2dx, 2dy and obtain the starting value for the decision parameter as
   $$P_0 = 2dx - dy$$
4. At each y_k along the line, starting at k=0 perform the following tests:
   If $P_k < 0$, the next point to plot is (x_k, y_k+1)
   and $P_{k+1} = P_k + 2dx$
   Otherwise, the next point to plot is (x_k+1, y_k+1)
   and $P_{k+1} = P_k + 2dx - 2dy$
5. Repeat step 4 dy times.

Advantages of Bresenham Line Drawing Algorithm

- An fast incremental algorithm
- Uses only integer calculations

Bresenham Circle Drawing Algorithm

Let us assume we have previously plotted a point (x_i, y_i).
The next point to be choosen is in between $N(x_i+1, y_i)$ and $S(x_i+1, y_i-1)$
this algorithm will find the distance of N, S from the circle
Let these distances are f(N) and f(S).



It will calculate the error in $f(S)$ and $f(N)$. Whichever will have less error will be selected.

We have equⁿ of the circle

$$x^2 + y^2 = r^2$$

So we have function

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

Now $f(N) = f(x_i+1, y_i) = (x_i+1)^2 + (y_i)^2 - r^2$

and $f(S) = f(x_i+1, y_i-1) = (x_i+1)^2 + (y_i-1)^2 - r^2$

Let us define the decision parameter, $d_i = f(N) + f(S)$

As the point N is outside the circle, the distance of N from the circle i.e. $f(N)$ will be positive.

Similarly as the point S is inside the circle, the distance of S from the circle i.e. $f(S)$ is negative.

if $d_i <= 0$: the term $f(S)$ is dominating ⇒ Point N is selected

$$\Rightarrow x_{i+1} = x_i + 1 \text{ and } y_{i+1} = y_i$$

if $d_i > 0$: the term $f(N)$ is dominating ⇒ Point S is selected

$$\Rightarrow x_{i+1} = x_i + 1 \text{ and } y_{i+1} = y_i - 1$$

Now $d_i = 2(x_i+1)^2 + y_i^2 + (y_i-1)^2 - 2r^2$

Similarly $d_{i+1} = 2(x_{i+1}+1)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2r^2$

$$d_{i+1} - d_i = 2[(x_{i+1}+1)^2 - (x_i+1)^2] + y_{i+1}^2 - y_i^2 + (y_{i+1}-1)^2 - (y_i-1)^2$$

$$\Rightarrow d_{i+1} = d_i + 2[(x_i+1+1)^2 - (x_i+1)^2] + y_{i+1}^2 - y_i^2 + (y_{i+1}-1)^2 - (y_i-1)^2$$

$$= d_i + 2(2x_i+3) + y_{i+1}^2 - y_i^2 + (y_{i+1}-1)^2 - (y_i-1)^2$$

For $d_i <= 0$ $\quad y_{i+1} = y_i$

So $d_{i+1} = d_i + 2(2x_i+3) + 0 + 0$

$$= d_i + 4x_i + 6$$

For $d_i > 0$ $\quad y_{i+1} = y_i - 1$

So $d_{i+1} = d_i + 2(2x_i+3) + (y_i-1)^2 - y_i^2 + (y_i-1-1)^2 - (y_i-1)^2$

$$= d_i + 4x_i + 6 + (y_i - 1 + y_i)(y_i - 1 - y_i) + (y_i - 2 + y_i - 1)(y_i - 2 - y_i + 1)$$

$$= d_i + 4x_i + 6 + 1 - 2y_i - 2y_i + 3$$

$$= d_i + 4(x_i - y_i) + 10$$

Calculation of Initial Decision Parameter

Initially $x=0$ and $y=r$

We have $d_i = 2(x_i+1)^2 + y_i^2 + (y_i-1)^2 - 2r^2$

Putting $x_i = 0$ and $y_i = r$

$$d_0 = 2 + r^2 + (r-1)^2 - 2r^2 = 3 - 2r$$

Algorithm

Step 1 : Obtain the radius of the circle r

Step 2 : Set the base decision parameter $d = 3-2r$

Step 3 : Set the base values of the coordinates $x=0$ and $y=r$

Step 4 : Compute the next pixels on the circle and update the decision factor

```
while (x <= y)
    plot(x,y)
    if(d <0) then
        d = d+4x+6
    else
        d = d+4(x-y)+10
        y = y-1
    Endif
    x = x+1
End while
```

Step 5 : Finish

Example : Draw the circle centered at (0,0) with radius 10

Solu^n^ $r = 10$, $d = 3-2.r = 3-2.10 = -17$

$(x,y) = (0,r) = (0,10)$

- Plot (0,10) . As $d=-17<0$, $d = d+4.0+6 = -11$, $x=1$
- Plot (1,10) . $d=-11<0$ so $d = -11+4.1+6 = -1$, $x=2$
- Plot (2,10) . $d=-1<0$ so $d = -1+4.2+6 = 13$, $x=3$
- Plot (3,10) . $d=13>0$ So $d = 13+4(3-10)+10 = -5$, $y=9$, $x=4$
- Plot (4,9) . $d=-5<0$ so $d=-5+4.4+6=17$, $x=5$
- Plot (5,9). $d=5>0$ so $d = 17+4(-4)+10 = 11$, $y=8$, $x=6$
- Plot (6,8). $d=11>0$ So $d = 11+4.(-2)+10=13$, $y=7$, $x=7$.
- Plot (7,7) $d=13>0$ So $d = 13+4(7-7)+10=23$ ∴ $y=6$, $x=8$

Now $x<=y$ not satisfied so stop.

Mid-Point Circle Drawing Algorithm

The equation for a circle is

$$x^2+y^2=r^2$$

where r is the radius of the circle whose center is at (0,0)

From this equation we have

$$y=\pm\sqrt{r^2-x^2}$$

From this we can have

$$y_0=\sqrt{20^2-0^2}\simeq 20$$
$$y_1=\sqrt{20^2-1^2}\simeq 20$$
$$y_2=\sqrt{20^2-2^2}\simeq 20$$
$$\vdots$$
$$y_{19}=\sqrt{20^2-19^2}\simeq 6$$
$$y_{20}=\sqrt{20^2-20^2}\simeq 0$$

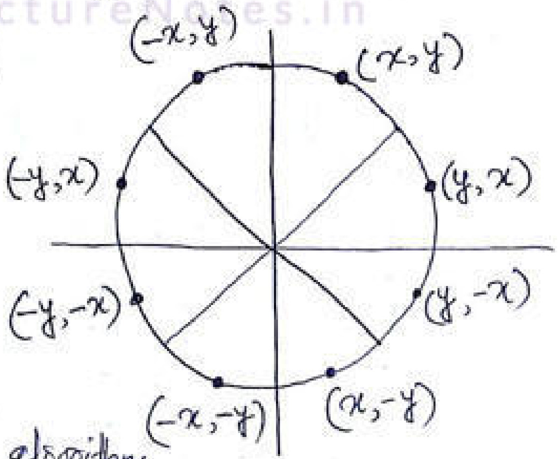
This is not the efficient way of finding the value of y at unit x intervals because of the following reasons:

- The resulting circle has large gaps
- The calculations are not very efficient as it has square (multiply) and square root operation.

Mid-Point Circle drawing is an efficient algorithm.

To make circle drawing algorithm more efficient we have to center the circle at (0,0). By doing this the circle will have eight-way symmetry

In mid-point circle algorithm we use eight-way symmetry so only ever calculate the points for the top right eighth of a circle, and then use symmetry to get the rest of the points

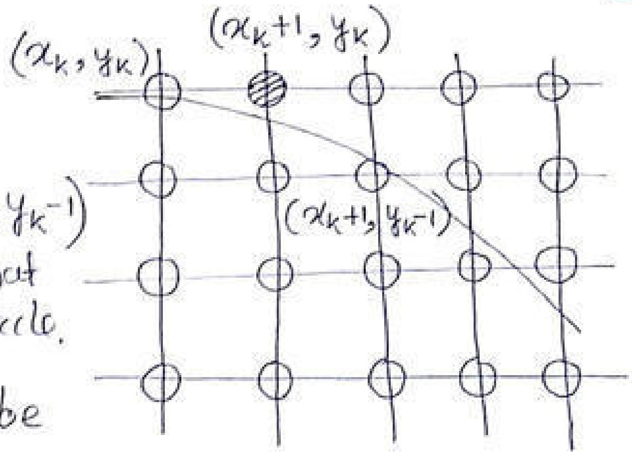


The mid-point circle drawing algorithm is developed by Jack Bresenham.

Assume that we have just plotted point (x_k, y_k)

The next point is a choice between (x_k+1, y_k) and (x_k+1, y_k-1)

We like to choose the point that is nearest to the actual circle.



The equation of the circle be written as

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

The equation evaluates as follows:

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

By evaluating this function at the midpoint between (x_k+1, y_k) and (x_k+1, y_k-1) we can make our decision

The decision variable can be defined as:

$$P_k = f\left(x_k+1, y_k-\tfrac{1}{2}\right)$$
$$= (x_k+1)^2 + \left(y_k-\tfrac{1}{2}\right)^2 - r^2$$

If $P_k < 0$ the mid point $(x_k+1, y_k-\frac{1}{2})$ is inside the circle i.e) (x_k+1, y_k) is closer to the circle

Else the mid point $(x_k+1, y_k-\frac{1}{2})$ is outside the circle i.e) (x_k+1, y_k-1) is closer

Similarly $P_{k+1} = (x_{k+1}+1)^2 + \left(y_{k+1}-\tfrac{1}{2}\right)^2 - r^2$

$$= (x_k+1+1)^2 + \left(y_{k+1}-\tfrac{1}{2}\right)^2 - r^2$$

$$P_{k+1} - P_k = (x_k+1+1)^2 - (x_k+1)^2 + \left(y_{k+1}-\tfrac{1}{2}\right)^2 - \left(y_k-\tfrac{1}{2}\right)^2$$
$$= (x_k+1)^2 + 2(x_k+1) + 1 - (x_k+1)^2 + \left(y_{k+1}^2 - y_{k+1} + \tfrac{1}{4}\right) - \left(y_k^2 - y_k + \tfrac{1}{4}\right)$$

$$\Rightarrow P_{k+1} = P_k + 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

Case 1 $P_k < 0$. So $y_{k+1} = y_k$

$$P_{k+1} = P_k + 2x_{k+1} + 1$$

Case 2 $P_k > 0$ So $y_{k+1} = y_k - 1$

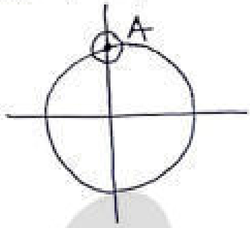
$$P_{k+1} = P_k + 2x_{k+1} + 1 - 2y_k + 1 + 1 = P_k + 2x_{k+1} - 2(y_k - 1) + 1$$
$$= P_k + 2x_{k+1} + 1 - 2y_{k+1}$$

Calculation of Initial Decision Variable

We have $P_k = f(x_k + 1, y_k - \frac{1}{2})$

Putting $k = 0$, $P_0 = f(x_0 + 1, y_0 - \frac{1}{2}) = f(0+1, r - \frac{1}{2})$

because we will start from the point A having coordinate $(0, r)$



So $P_0 = f(1, r - \frac{1}{2})$
$$= 1 + (r - \tfrac{1}{2})^2 - r^2$$
$$= 1 + r^2 - r + \tfrac{1}{4} - r^2$$
$$= \tfrac{5}{4} - r \simeq 1 - r$$

Algorithm

Step 1 Input radius r and circle centre (x_c, y_c), then set the coordinates for the first point on the circumference of a circle centered on the origin as:
$(x_0, y_0) = (0, r)$

Step 2 Calculate the initial value of the decision parameter as:
$$P_0 = 5/4 - r \simeq 1 - r$$

Step 3 Starting with $k = 0$ at each position x_k perform the following tests:

If $P_k < 0$, the next point along the circle centered on $(0,0)$ is $(x_k + 1, y_k)$ and
$$P_{k+1} = P_k + 2x_{k+1} + 1$$

Else the next point along the circle is $(x_k + 1, y_k - 1)$ and
$$P_{k+1} = P_k + 2x_{k+1} + 1 - 2y_{k+1}$$

Step 4 Determine symmetry points in the other seven points

Step 5 Move each calculated pixel position (x, y) onto the circular path centered at (x_c, y_c) to plot the coordinate values:

$$x = x + x_c \qquad y = y + y_c$$

Step 6 Repeat steps 3 to 5 until $x >= y$ not satisfied
(Repeat while $x < y$)

Example Draw a circle centered at (0,0) with radius 10

Solu^n^ $(x_0, y_0) = (0, 10)$

$P_0 = \frac{5}{4} - 10 \simeq 1 - 10 = -9$

k=0 So next point to be plotted is (1, 10) since $P_0 < 0$

$P_1 = P_0 + 2.1 + 1 = -9 + 2 + 1 = -6$

k=1 Next point is (2, 10) since $P_1 < 0$

$P_2 = P_1 + 2.2 + 1 = -6 + 4 + 1 = -1$

k=2 Next point is (3, 10) since $P_2 < 0$

$P_3 = P_2 + 2.3 + 1 = -1 + 6 + 1 = 6$

k=3 Next point is (4, 9) since $P_3 > 0$

$P_4 = P_3 + 2.4 + 1 - 2.9 = 6 + 8 + 1 - 18 = -3$

k=4 Next point is (5, 9) since $P_4 < 0$

$P_5 = P_4 + 2.5 + 1 = -3 + 10 + 1 = 8$

k=5 Next point is (6, 8) since $P_5 > 0$

$P_6 = P_5 + 2.6 + 1 - 2.8 = 8 + 12 + 1 - 16 = 5$

k=6 Next point is (7, 7) since $P_6 > 0$

$P_7 = P_6 + 2.7 + 1 - 2.7 = 5 + 1 = 6$

Now $x >= y$ So stop

Note that for each of the points (1, 10), (2, 10), ... (7, 7) determine ~~their~~ its symmetry points & plot.

Advantage

Eight-way symmetry can hugely reduce the work in drawing a circle

Comparison of Bresenham Circle Drawing & Midpoint Circle Drawing

- Bresenham algorithm results in a much more smoother circle compared to midpoint circle algorithm.
- In mid point, decision parameter depends on previous decision parameter and corresponding pixels whereas in Bresenham decision parameter only depends on previous decision parameter.

Computer Graphics

Topic:

Two Dimensional Geometric Transformation

Contributed By:

Jasaswi Prasad Mohanty

Two Dimensional Transformations

- Transformation means a change in either position or size or shape or orientation of any graphical object.
- When a transformation takes place on a 2D plane it is called 2D transformation.
- Different types of transformations are:
  1. Translation
  2. Scaling
  3. Rotation
  4. Reflection
  5. Shearing

Translation

- A translation is applied to an object by repositioning it along a straight-line path from one coordinate location to another.
- We translate a two-dimensional point by adding translation distances, t_x and t_y to the original coordinate position (x, y) to move the point to a new position (x', y')

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

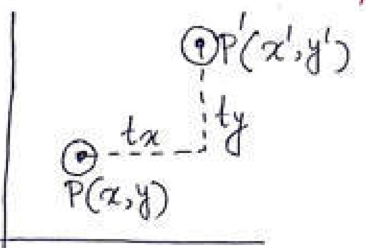
- The translation distance pair (t_x, t_y) is called a translation vector or shift vector.
- In matrix form it is represented as the following transformation equation:

$$P' = P + T$$

where $P = \begin{bmatrix} x \\ y \end{bmatrix}$, $P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$, $T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$: translation matrix

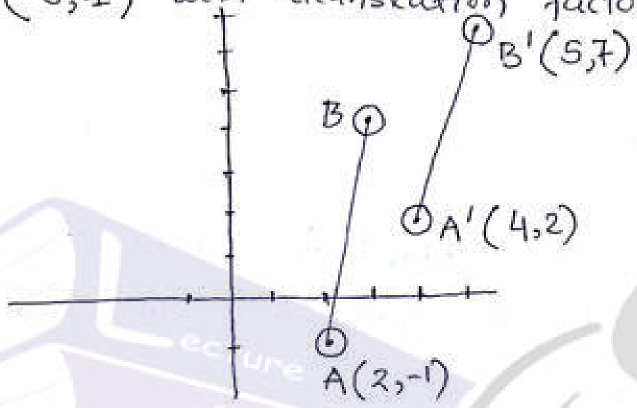
- Translation is a rigid-body transformation that moves objects without deformation.
- A ~~tran~~ straight line segment is translated by applying the transformation equation to each of the line end points and redrawing the line between new end points.

- Polygons are translated by adding the translation vector to the coordinate position of each vertex and regenerating the polygon using new set of vertex coordinates.



- To translate a circle or ellipse we translate the center coordinates and redraw the circle or ellipse in the new center.

__Example__ Translate the line between points A(2,-1) and B(3,4) with translation factors $tx = 2$, $ty = 3$



$$A' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \end{bmatrix} + \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

$$B' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \end{bmatrix}$$

Scaling

- A scaling transformation alters the size of an object
- In the scaling process either we expand or compress the dimension of the object
- Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.
- Let P(x,y) be the original coordinates, the scaling factors are Sx : scaling factor along x-axis
  Sy : scaling factor along y-axis

  and the produced coordinates are (x', y')

  Mathematically $x' = x . Sx$ and $y' = y . Sy$)

If we denote (x, y) by P and (x', y') by P' then

$$P' = P \cdot S$$

In matrix form

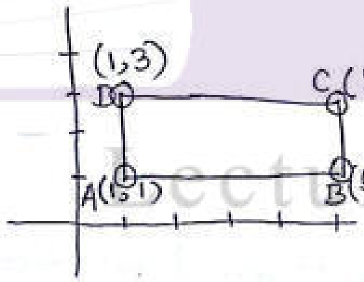
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

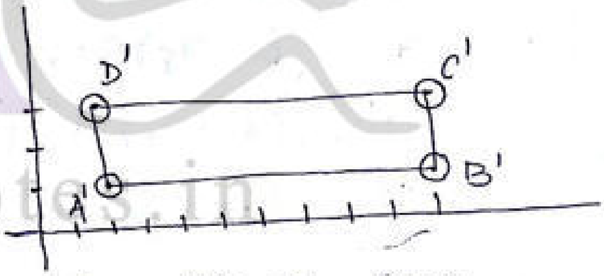
NOTE:

1) Any positive numeric value can be assigned to the scaling factor S_x and S_y. Values less than 1 reduce the size of objects whereas values greater than 1 produces an enlargement.
2) When $S_x = S_y$ the scaling is a uniform scaling.
3) When $S_x \neq S_y$ the scaling is a differential scaling
4) The above scaling is done with respect to origin.

Example A rectangle is defined as $\begin{bmatrix} 1 & 5 & 5 & 1 \\ 1 & 1 & 3 & 3 \end{bmatrix}$

Scale the rectangle with scaling factor $S_x = 2$ and $S_y = 1$





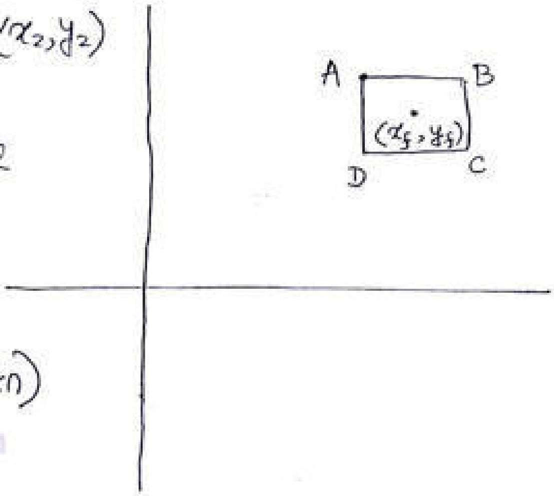
$$A' = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad B' = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 1 \end{bmatrix}$$

$$C' = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \end{bmatrix} = \begin{bmatrix} 10 \\ 3 \end{bmatrix}, \quad D' = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Scaling w.r.t. fixed point (x_f, y_f)

1. Translate the fixed point to the origin
2. Scale the object with respect to origin
3. Retranslate the scaling object to original position

Suppose we have a rectangle ABCD having coordinates $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$ & $D(x_4, y_4)$.
We want to scale the rectangle with scaling factor S_x and S_y with respect to point (x_f, y_f)



After step 1 (Translate to origin) the points are

$$A' = (x_1 - x_f, y_1 - y_f)$$
$$B' = (x_2 - x_f, y_2 - y_f)$$
$$C' = (x_3 - x_f, y_3 - y_f)$$
$$D' = (x_4 - x_f, y_4 - y_f)$$

After step 2 (scale with respect to origin)

$$A' = ((x_1 - x_f)S_x, (y_1 - y_f)S_y)$$
$$B' = ((x_2 - x_f)S_x, (y_2 - y_f)S_y)$$
$$C' = ((x_3 - x_f)S_x, (y_3 - y_f)S_y)$$
$$D' = ((x_4 - x_f)S_x, (y_4 - y_f)S_y)$$

After step 3 (Retranslate to the original position)

$$A' = ((x_1 - x_f)S_x + x_f, (y_1 - y_f)S_y + y_f)$$
$$B' = ((x_2 - x_f)S_x + x_f, (y_2 - y_f)S_y + y_f)$$
$$C' = ((x_3 - x_f)S_x + x_f, (y_3 - y_f)S_y + y_f)$$
$$D' = ((x_4 - x_f)S_x + x_f, (y_4 - y_f)S_y + y_f)$$

In general
$$x' = (x - x_f)S_x + x_f = x.S_x + x_f(1 - S_x)$$
$$y' = (y - y_f)S_y + y_f = y.S_y + y_f(1 - S_y)$$

In matrix format

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1 - S_x & 0 \\ 0 & 1 - S_y \end{bmatrix} \begin{bmatrix} x_f \\ y_f \end{bmatrix}$$

NOTE: The fixed point can be any point inside or outside of the object

Problem Scale the rectangle A(1,1), B(5,1), C(5,3), D(1,3) with fixed point (5,2) and scaling factor $S_x = 2, S_y = 2$

Soln

$$A' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1-2 & 0 \\ 0 & 1-2 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ 2 \end{bmatrix} + \begin{bmatrix} -5 \\ -2 \end{bmatrix} = \begin{bmatrix} -3 \\ 0 \end{bmatrix}$$

$$B' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 5 \\ 1 \end{bmatrix} + \begin{bmatrix} 1-2 & 0 \\ 0 & 1-2 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 10 \\ 2 \end{bmatrix} + \begin{bmatrix} -5 \\ -2 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

$$C' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \end{bmatrix} + \begin{bmatrix} 1-2 & 0 \\ 0 & 1-2 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 10 \\ 6 \end{bmatrix} + \begin{bmatrix} -5 \\ -2 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$$

$$D' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} + \begin{bmatrix} 1-2 & 0 \\ 0 & 1-2 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ 6 \end{bmatrix} + \begin{bmatrix} -5 \\ -2 \end{bmatrix} = \begin{bmatrix} -3 \\ 4 \end{bmatrix}$$

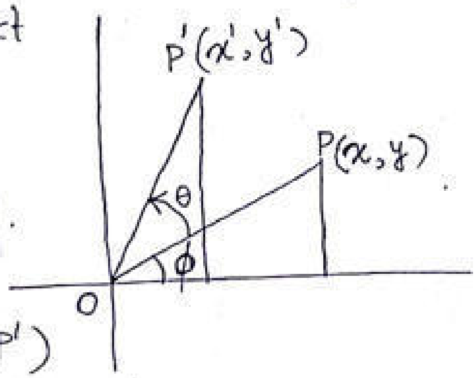
Rotation

- To generate a rotation, we have to specify a rotation angle θ and the position (x_r, y_r) of the rotation point (pivot point) about which the object is to be rotated.

- Possitive values for the rotation angle define counterclockwise rotations about the pivot point and negative values rotate objects in the clockwise direction.

- Let us discuss the rotation of a point P(x,y) when the pivot point is at the origin (0,0)

Suppose P is a point having coordinates (x, y) is rotated with an angle θ with respect to origin.



After rotation the point is P' having coordinates (x', y').

Let r is the constant distance of the points (P or P') from the origin.

ϕ is the original angular position of the point P from the horizontal (x-axis).

The polar coordinates of P can be expressed as

$$x = r\cos\phi$$
$$y = r\sin\phi$$

Similarly the polar coordinates of P' can be expressed as

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

In matrix format:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$ ← Rotation Matrix in anti-clockwise direction.

Similarly $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$ ← Rotation Matrix in clockwise direction.

Rotation about a fixed point (x_f, y_f)

To rotate a point $P(x, y)$ with respect to a fixed point (x_f, y_f) with an angle θ follow the following steps:

1. Bring the fixed point to the origin.
   The coordinates of P will be $(x - x_f, y - y_f)$
2. At origin rotate the object
   After rotation the coordinates of the point will be

$$(x - x_f)\cos\theta - (y - y_f)\sin\theta$$
$$(x - x_f)\sin\theta + (y - y_f)\cos\theta$$

3. Retranslate the rotated object to the original position.
After that the coordinates of P' are:

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

Homogeneous Coordinate System

Why do we use homogeneous coordinates?

1. We can represent all transformations as matrix multiplications.
2. We can capture composite transformations conveniently
3. We can represent a point at infinity which is needed when we want to represent a point at infinity in a certain direction.

To express any 2D transformation as a matrix multiplication, we represent each cartesian coordinate position (x, y) as homogeneous coordinate system like

$$\underset{\text{cartesian}}{(x, y)} = \underset{\text{homogeneous}}{(xh, yh, h)}$$

where h represents a homogeneous parameter which is a non-zero value such that

$$x = \frac{xh}{h}, \quad y = \frac{yh}{h}$$

For our convenience, we generally take $h = 1$

$$(x, y) \rightarrow (x, y, 1) \quad \text{or} \quad \begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Eg: $(0, 1, 2) = \left(\frac{0}{2}, \frac{1}{2}, \frac{2}{2}\right) \rightarrow (0, 1/2)$

$(0, 0.5, 1) \rightarrow (0, 0.5)$

Homogeneous co-ordinate representation of:

1. Translation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad \text{or} \quad P' = T(t_x, t_y) \cdot P$$

2. Scaling

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad \text{or} \quad P' = S(S_x, S_y) \cdot P$$

3. Rotation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad \text{or} \quad P' = R(\theta) \cdot P$$

Inverse Transformations

1. Translation

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

2. Scaling

$$S^{-1} = \begin{bmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. Rotation

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

Composite Transformations

- It is a matrix product of individual transformations
- Forming products of transformation matrices is often referred to as concatenation or composition of matrices.
- For column matrix representation of coordinate positions, we form composite transformations by multiplying matrices in order from right to left.

Translations

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

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

where P and P' are represented as homogeneous-coordinate column vectors

$$T(t_{x2} \cdot t_{y2}) \cdot T(t_{x1} \cdot t_{y1}) = \begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & t_{x1}+t_{x2} \\ 0 & 1 & t_{y1}+t_{y2} \\ 0 & 0 & 1 \end{bmatrix} = T(t_{x1}+t_{x2}, t_{y1}+t_{y2})$$

Therefore we can conclude <u>two successive translations are additive</u>

Rotation

Two successive rotations applied to point P produce the transformed position

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

So <u>two successive rotations are additive.</u>

Scaling

Two successive scaling operations to point P produce the following

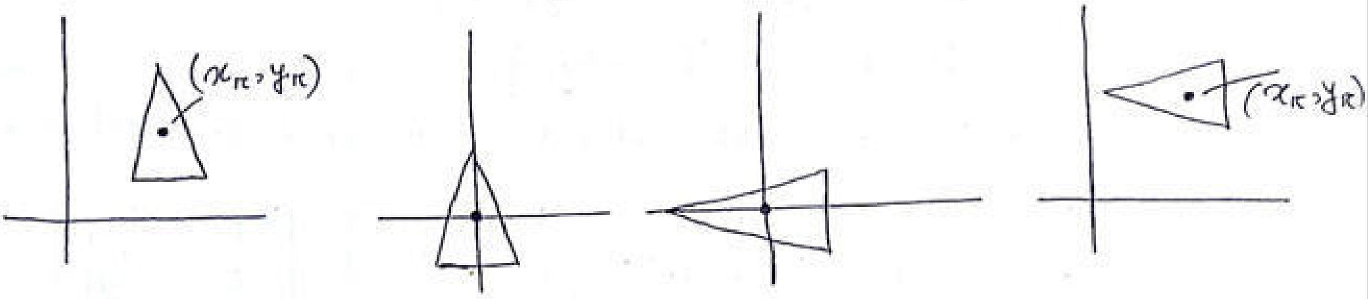
$$P' = S(S_{x2}, S_{y2}) \cdot \{S(S_{x1}, S_{y1}) \cdot P\} = \{S(S_{x2}, S_{y2}) \cdot S(S_{x1}, S_{y1})\} \cdot P$$
$$= S(S_{x1} \cdot S_{x2}, S_{y1} \cdot S_{y2}) \cdot P$$

Two successive scaling operations are multiplicative

General Pivot-Point Rotation

We can generate rotations about any selected pivot point (x_r, y_r) by performing the following sequence of operations:

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



Original Position of object and Pivot Point

Translation of object so that Pivot Point (x_r, y_r) is at origin

Rotation about origin

Translation of object so that the Pivot Point is returned to position (x_r, y_r)

The transformation, $T = T(x_r, y_r) \cdot R(\theta)\, T(-x_r, -y_r)$

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

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

T can be expressed as $R(x_r, y_r, \theta)$

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

General Fixed-Point Scaling

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

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

Concatenating the matrices for these three operations produces the required scaling matrix

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

or $T(x_f, y_f) \cdot S(s_x, s_y) \cdot T^{-1}(x_f, y_f) = S(x_f, y_f, s_x, s_y)$

General scaling Directions

- Parameters s_x and s_y scale objects along the x and y directions
- To scale an object in other directions first rotate the object to allign the desired scaling directions with the coordinate axes before applying the scaling transformations
- Without changing the orientation of the object, we first perform a rotation so that the directions for s_1 and s_2 coincide with the x and y axes respectively. Then the scaling transformation is applied followed by an opposite rotation.
- The composite matrix resulting from the product of these three transformations is

$$R^{-1}(\theta) \cdot S(s_1, s_2) \cdot R(\theta)$$

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

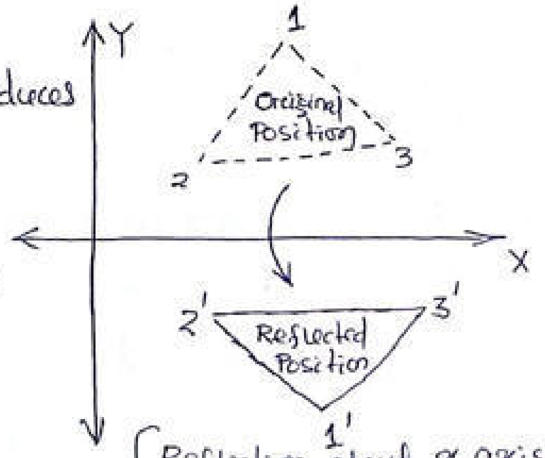
Concatenation Properties

- If we want to translate and rotate (as example of two transformations) we must be careful about the order in which the composite matrix is evaluated.
- For some special cases multiplication of transformation matrices is commutative.

  Eg: Two successive rotations ; Two successive translations, Two successive scaling, Rotation and uniform scaling.

Reflection

- It is a transformation that produces a mirror image of an object
- The mirror image for a 2D reflection is generated relative to an axis of reflection by rotating the object 180° about the reflection axis.

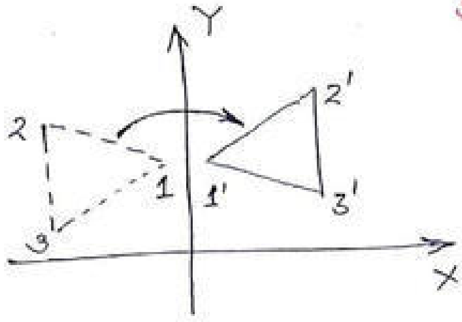


[Reflection about x axis]

Reflection about x-axis (y=0)

- Reflection about x-axis is accomplished with transformation matrix

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



- This transformation keeps x values same, but flips y values of the coordinate positions.

ie, $x' = x$
$y' = -y$

- Reflection is shown in the figure in the previous page.

Reflection about y-axis (x=0)

- Transformation Matrix

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

- This transformation keeps y values same but flips x values

ie, $x' = -x$
$y' = y$

Reflection about the origin

(Reflection of an object relative to an axis perpendicular to the xy plane and passing through the coordinate origin)

- Transformation Matrix

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

- This transformation flips both x and y values

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

Reflection about the line $y = x$

- Transformation Matrix

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

- This transformation interchanges the x, y values
  i.e. $x' = y$
  $y' = x$

Reflection about the line $y = -x$

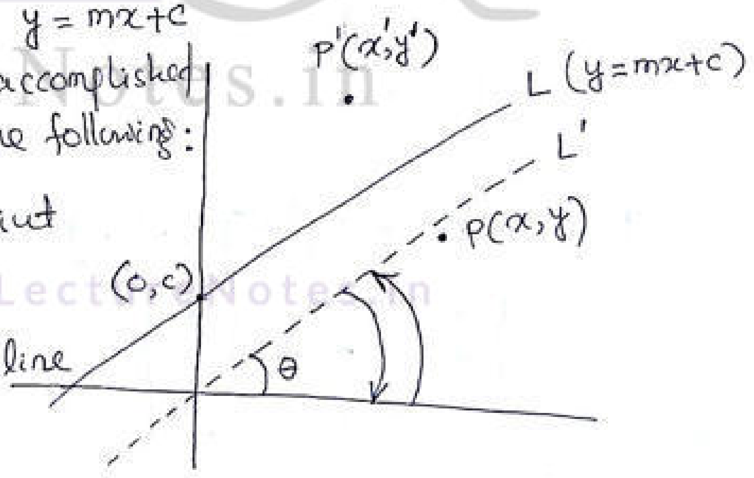
- Transformation Matrix

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

- This transformation interchanges and flips the x, y values
  i.e. $x' = -y$
  $y' = -x$

Reflection of an object about any arbitrary line $y = mx + c$

Reflection about any line $y = mx + c$ in the xy plane can be accomplished with a combination of the following:



1. Translate the fixed point to the origin
2. Rotate (clock-wise) the line L so that it coincides with x-axis
3. Reflect the ~~line~~ object about x-axis
4. Rotate (anti-clockwise) the line with the same angel
5. Retranslate the line L to the original position.

1. $T^{-1}_{(0,c)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -c \\ 0 & 0 & 1 \end{bmatrix}$ Here $t_x = 0$, $t_y = c$

2. $R^{-1}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ Clock wise Rotation

3. $T_{Ref(x)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

4. $R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ Anticlock wise Rotation

5 $T_{(0,c)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & c \\ 0 & 0 & 1 \end{bmatrix}$

So Transformation, $T = T(0,c) \cdot R(\theta) \cdot T_{Ref(x)} R^{-1}(\theta) \cdot T^{-1}(0,c)$

$$= \begin{bmatrix} \frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & \frac{-2cm}{m^2+1} \\ \frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & \frac{2c}{m^2+1} \\ 0 & 0 & 1 \end{bmatrix}$$

where $m = \tan\theta$

$\Rightarrow \sin\theta = \frac{m}{\sqrt{m^2+1}}$

$\cos\theta = \frac{1}{\sqrt{m^2+1}}$

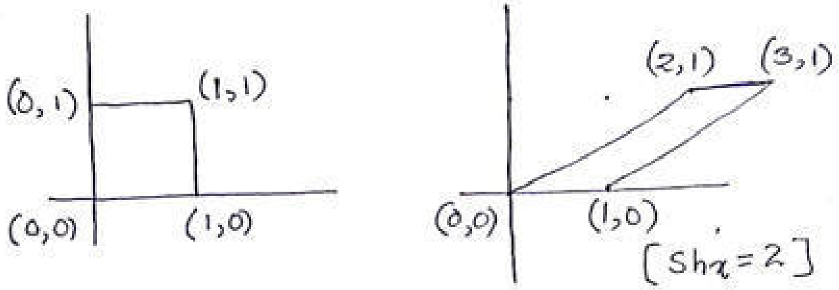
The new position of point $P(x,y)$,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shearing Transformation

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 a shear.

Shearing along x-directions



An x-direction shear relative to the x-axis is produced with transformation matrix

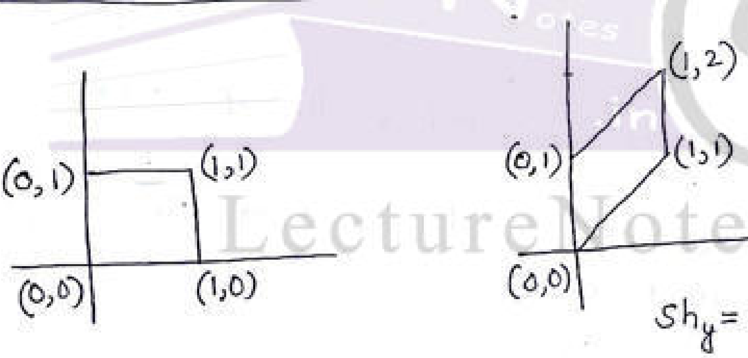
$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

sh_x : shearing factor along x-axis.

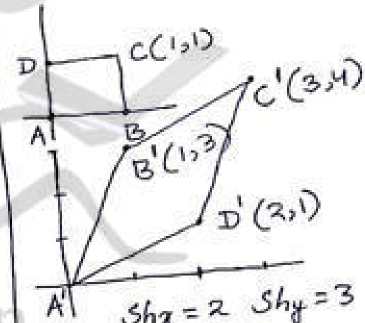
which transforms coordinate positions as

$$x' = x + sh_x \cdot y, \quad y' = y$$

Shearing along y-directions



Shearing along x&y directions



$Sh_x = 2 \quad Sh_y = 3$

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

An y-direction shear relative to the y-axis is produced with transformation matrix

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

Sh_y : Shearing factor along y-axis.

which transforms coordinate positions as

$$x' = x, \quad y' = y + Sh_y \cdot x$$

NOTE: 1) A coordinate position (x, y) is shifted horizontally / vertically by an amount proportional to its distance from the x-axis / y-axis depending on Sh_x / Sh_y.

2) Negative values for sh_x/sh_y shift coordinate positions to the left/down of y-axis/x-axis.

Shearing along x-directions relative to other reference lines

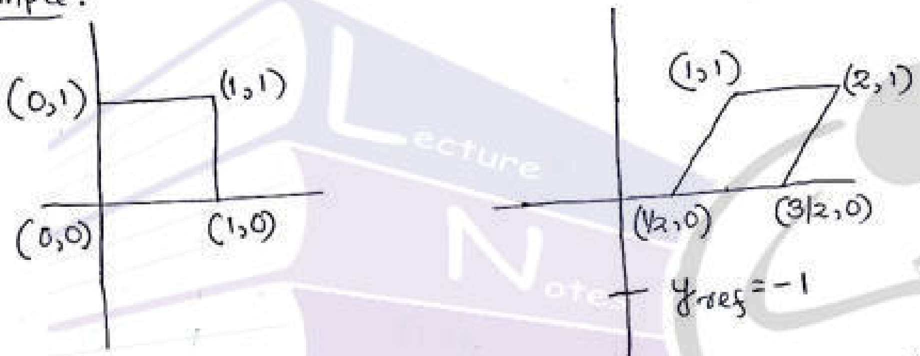
An x-direction shear relative to the line $y = y_{ref}$ is produced with reference matrix

$$\begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

with coordinate positions transformed as

$$x' = x + sh_x(y - y_{ref}) , \quad y' = y$$

Example:



$Sh_x = 1/2$
$y_{ref} = -1$

Shearing along y-directions relative to other reference lines

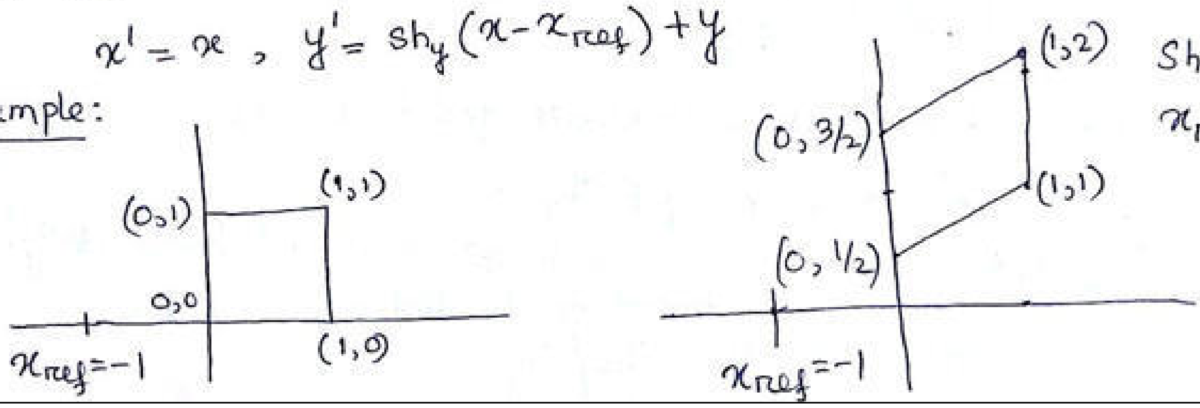
An y-direction shear relative to the line $x = x_{ref}$ is produced reference matrix

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

with coordinate positions transformed as

$$x' = x , \quad y' = sh_y(x - x_{ref}) + y$$

Example:



$Sh_y = 1/2$
$x_{ref} = -1$

Translation $T(t_x, t_y)$	$\begin{bmatrix}1 & 0 & t_x\\0 & 1 & t_y\\0 & 0 & 1\end{bmatrix}$
Scaling $S(s_x, s_y)$	$\begin{bmatrix}s_x & 0 & 0\\0 & s_y & 0\\0 & 0 & 1\end{bmatrix}$
Rotation anticlockwise $R(\theta)$	$\begin{bmatrix}\cos\theta & -\sin\theta & 0\\\sin\theta & \cos\theta & 0\\0 & 0 & 1\end{bmatrix}$
Inverse Translation $T^{-1}(t_x, t_y)$	$\begin{bmatrix}1 & 0 & -t_x\\0 & 1 & -t_y\\0 & 0 & 1\end{bmatrix}$
Inverse Scaling $S^{-1}(s_x, s_y)$	$\begin{bmatrix}1/s_x & 0 & 0\\0 & 1/s_y & 0\\0 & 0 & 1\end{bmatrix}$
Rotation clockwise $R^{-1}(\theta)$	$\begin{bmatrix}\cos\theta & \sin\theta & 0\\-\sin\theta & \cos\theta & 0\\0 & 0 & 1\end{bmatrix}$
General Fixed-Point (x_f, y_f) Scaling $S(x_f, y_f, s_x, s_y)$	$\begin{bmatrix}s_x & 0 & x_f(1-s_x)\\0 & s_y & y_f(1-s_y)\\0 & 0 & 1\end{bmatrix}$
General Pivot-Point (x_r, y_r) anticlockwise Rotation $R(x_r, y_r, \theta)$	$\begin{bmatrix}\cos\theta & -\sin\theta & x_r(1-\cos\theta)+y_r\sin\theta\\\sin\theta & \cos\theta & y_r(1-\cos\theta)-x_r\sin\theta\\0 & 0 & 1\end{bmatrix}$
Scaling in other directions $S(s_1, s_2, \theta)$	$\begin{bmatrix}s_1\cos^2\theta + s_2\sin^2\theta & (s_2-s_1)\cos\theta\sin\theta & 0\\(s_2-s_1)\cos\theta\sin\theta & s_1\sin^2\theta + s_2\cos^2\theta & 0\\0 & 0 & 1\end{bmatrix}$
Reflection about x-axis	$\begin{bmatrix}1 & 0 & 0\\0 & -1 & 0\\0 & 0 & 1\end{bmatrix}$
Reflection about y-axis	$\begin{bmatrix}-1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$

Reflection about the origin (Reflection relative to an axis perpendicular to the xy-plane and passing through origin)	$\begin{bmatrix}-1 & 0 & 0\\0 & -1 & 0\\0 & 0 & 1\end{bmatrix}$
Reflection about the line y=x	$\begin{bmatrix}0 & 1 & 0\\1 & 0 & 0\\0 & 0 & 1\end{bmatrix}$
Reflection about the line y=−x	$\begin{bmatrix}0 & -1 & 0\\-1 & 0 & 0\\0 & 0 & 1\end{bmatrix}$
Reflection about any arbitrary line y=mx+c	$\begin{bmatrix}\frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & \frac{-2cm}{m^2+1}\\\frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & \frac{2c}{m^2+1}\\0 & 0 & 1\end{bmatrix}$
Shearing along x-directions	$\begin{bmatrix}1 & sh_x & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$
Shearing along y-directions	$\begin{bmatrix}1 & 0 & 0\\sh_y & 1 & 0\\0 & 0 & 1\end{bmatrix}$
Shearing along x-directions relative to other reference line $y=y_{ref}$	$\begin{bmatrix}1 & sh_x & -sh_x y_{ref}\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$
Shearing along y-directions relative to other reference line $x=x_{ref}$	$\begin{bmatrix}1 & 0 & 0\\sh_y & 1 & -sh_y x_{ref}\\0 & 0 & 1\end{bmatrix}$
Shearing along x and y directions	$\begin{bmatrix}1 & sh_x & 0\\sh_y & 1 & 0\\0 & 0 & 1\end{bmatrix}$

Problems on 2-D Transformations

1. Perform a 45° rotation of triangle $A(0,0)$, $B(1,1)$, $C(5,2)$
   a) about the origin
   b) about $P(-1,-1)$

Solu^n

a)
$$[A'\ B'\ C'] = R(45^\circ)[A\ B\ C] = \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 3/\sqrt{2} \\ 0 & 2/\sqrt{2} & 7/\sqrt{2} \\ 1 & 1 & 1 \end{bmatrix}$$

b)
$$[A'\ B'\ C'] = R(-1,-1,45^\circ).[A\ B\ C]$$

$$= \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & -1(1-\cos 45^\circ)+(-1).\sin 45^\circ \\ \sin 45^\circ & \cos 45^\circ & -1(1-\cos 45^\circ)-(-1)\sin 45^\circ \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & -1 \\ 1/\sqrt{2} & 1/\sqrt{2} & \sqrt{2}-1 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & -1 & 3/\sqrt{2}-1 \\ \sqrt{2}-1 & 2\sqrt{2}-1 & 9/\sqrt{2}-1 \\ 1 & 1 & 1 \end{bmatrix}$$

2. Magnify the triangle with vertices $A(0,0)$, $B(1,1)$, $C(5,2)$ to twice its size while keeping $C(5,2)$ fixed.

Magnifying the Δ ABC keeping C fixed is same as magnifying about the fixed point C

$$[A'\ B'\ C'] = \begin{bmatrix} S_x & 0 & x_f(1-S_x) \\ 0 & S_y & y_f(1-S_y) \\ 0 & 0 & 1 \end{bmatrix}.[A\ B\ C] = \begin{bmatrix} 2 & 0 & 5(1-2) \\ 0 & 2 & 2(1-2) \\ 0 & 0 & 1 \end{bmatrix}.\begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 0 & -5 \\ 0 & 2 & -2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -5 & -3 & 5 \\ -2 & 0 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

3. Reflect the diamond-shaped polygon whose vertices are A(-1,0), B(0,-2), C(1,0) and D(0,2) about
   a) the horizontal line $y=2$
   b) the vertical line $x=2$
   c) the line $y = x+2$

Solu^n

a) The equation of the line is $y=2$

Here $m = 0$ & $c = 2$

$$[A' \ B' \ C' \ D'] = \begin{bmatrix} \frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & -\frac{2cm}{m^2+1} \\ \frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & \frac{2c}{m^2+1} \\ 0 & 0 & 1 \end{bmatrix} [A \ B \ C \ D]$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 4 & 6 & 4 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

b) The vertical line $x=2$ has no y intercept and an infinite loop. So we translate the line $x=2$ two units over to the y-axis then reflect about y-axis and finally retranslate back two units.

$$[A' \ B' \ C' \ D'] = T^{-1}(2,0) . T_{Ref(y)} . T(2,0) . [A \ B \ C \ D]$$

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

$$= \begin{bmatrix} -1 & 0 & -4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1-4 & -4 & -1-4 & -4 \\ 0 & -2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -3 & -4 & -5 & -4 \\ 0 & -2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

c) The line $y = x + 2$ has slope, $m = 1$ and y intercept $c = 2$

$$[A' \ B' \ C' \ D'] = \begin{bmatrix} \frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & -\frac{2cm}{m^2+1} \\ \frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & \frac{2c}{m^2+1} \\ 0 & 0 & 1 \end{bmatrix} [A \ B \ C \ D]$$

$$= \begin{bmatrix} 0 & 1 & -2 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & -4 & -2 & 0 \\ 1 & 2 & 3 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

4. The matrix $\begin{bmatrix} 1 & b \\ a & 1 \end{bmatrix}$ defines a transformation called a simultaneous shearing or shearing for short.

The special case when $b=0$ is called shearing in the x direction. When $a=0$, we have shearing in the y direction. Illustrate the effect of these shearing transformations on the square $A(0,0)$, $B(1,0)$, $C(1,1)$ and $D(0,1)$ when $a=2$ & $b=3$

$$[A' \ B' \ C' \ D'] = SH(sh_x, sh_y) . [A \ B \ C \ D]$$

$$= \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} [A \ B \ C \ D] = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 3 & 2 \\ 0 & 3 & 4 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

5. Find the instance transformation which places a half-size copy of the square $A(0,0)$, $B(1,0)$, $C(1,1)$, $D(0,1)$ into a master picture coordinate system so that the center of the square is at $(-1,-1)$.

The centre of the square ABCD is at $P(\frac{1}{2}, \frac{1}{2})$. We shall first apply a scaling transformation while keeping P fixed. Then we shall apply a translation that moves $P(\frac{1}{2}, \frac{1}{2})$ to $P'(-1,-1)$

$[A', B', C', D'] = T(-3/2, -3/2) \cdot S(1/2, 1/2, 1/2, 1/2) \cdot [A\ B\ C\ D]$

$$[A'\ B'\ C'\ D'] = \begin{bmatrix} 1 & 0 & -3/2 \\ 0 & 1 & -3/2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & \frac{1}{2}(1-\frac{1}{2}) \\ 0 & 1/2 & 1/2(1-1/2) \\ 0 & 0 & 1 \end{bmatrix} [A\ B\ C\ D]$$

$$= \begin{bmatrix} 1/2 & 0 & -5/4 \\ 0 & 1/2 & -5/4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -5/4 & -3/4 & -3/4 & -5/4 \\ -5/4 & -5/4 & -3/4 & -3/4 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

6. Prove that a shear transformation can be expressed in terms of rotation and scaling operations.

Soluⁿ The general shear matrix for x and y components is given as

$$T_{sh} = \begin{bmatrix} 1 & Sh_x & 0 \\ Sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ where Sh_x and Sh_y are shearing factors respectively. —— (i)

The scaling and rotation matrices are given as

$$T_S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } T_R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The combined transformation in sequence is equivalent to product of scaling & rotation

$$T = T_S \cdot T_R = \begin{bmatrix} S_x\cos\theta & -S_x\sin\theta & 0 \\ S_y\sin\theta & S_y\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{—— (ii)}$$

comparing (i) & (ii) we have

$$\left.\begin{aligned} S_x\cos\theta &= 1 \\ S_y\cos\theta &= 1 \\ Sh_x &= -S_x\sin\theta \\ Sh_y &= S_y\sin\theta \end{aligned}\right\} \quad \text{—— (iii)}$$

or $S_x = 1/\cos\theta$ & $S_y = 1/\cos\theta$ —— (iv)

Substituting S_x & S_y in (iii)

$$\left.\begin{aligned} Sh_x &= -S_x\sin\theta = -\frac{1}{\cos\theta}\cdot\sin\theta = -\tan\theta \\ Sh_y &= S_y\sin\theta = \frac{1}{\cos\theta}\cdot\sin\theta = \tan\theta \end{aligned}\right\} \quad \text{—— (v)}$$

Therefore using (iii) and (v), the shear transformation matrix can be expressed in terms of scaling & rotation given by

$$T_{sh} = \begin{bmatrix} 1 & -\tan\theta & 0 \\ \tan\theta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where θ is the angle of rotation, S_x & S_y are scaling factors in x and y directions such that

$S_x \cos\theta = S_y \cos\theta = 1$

Computer Graphics

Topic:

Two Dimensional Viewing

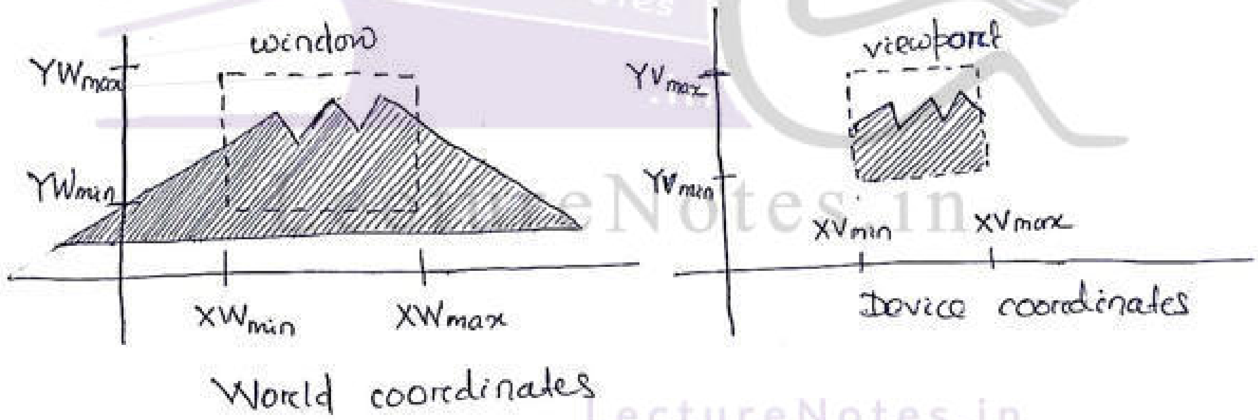
Contributed By:

Jasaswi Prasad Mohanty

Two-Dimensional Viewing

- A graphics package allows a user to specify which part of a defined picture to be displayed and where that part is to be placed on the display device using a procedure known as clipping.
- Any convenient cartesian coordinate system, referred to as the world-coordinate reference frame can be used to define the picture.
- For a two-dimensional picture, a view is selected by specifying a subarea of the total picture area.
- The picture parts within the selected areas are then mapped onto specified areas of the device coordinates.
- Transformations from world to device coordinates involve translation, rotation, and scaling operations, as well as procedures for deleting those parts of the picture that are outside the limits of a selected display area.

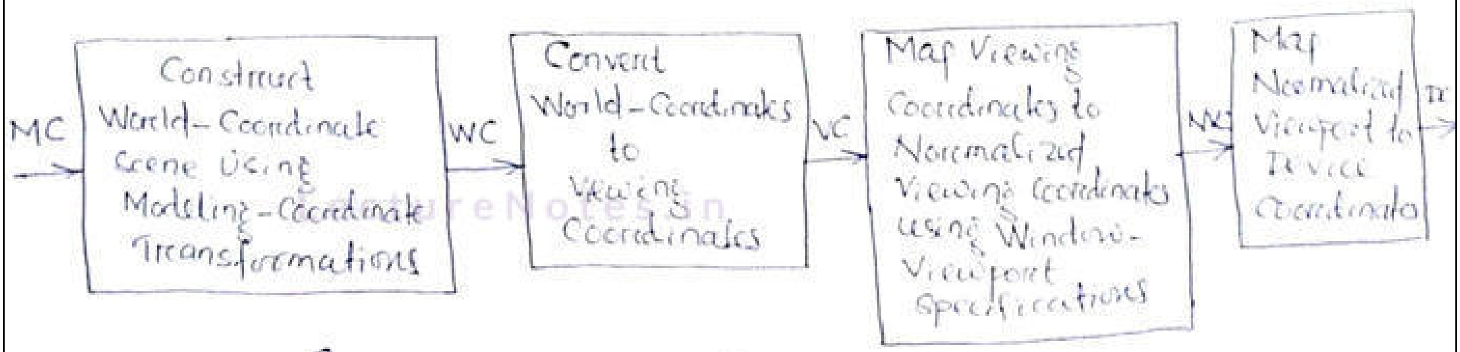
Viewing Pipeline



- A world-coordinate area selected for display is called a window. The objects within the window are displayed only.
- An area on a display device to which a window is mapped is called a viewport.
- The window defines what is to be viewed; the viewport defines where it is to be displayed. Both are rectangulars normally.
- Mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation.

- The two-dimensional viewing transformation is simply referred to as the window-to-viewport transformation or the windowing transformation.

- The viewing transformation is carried out in several steps:



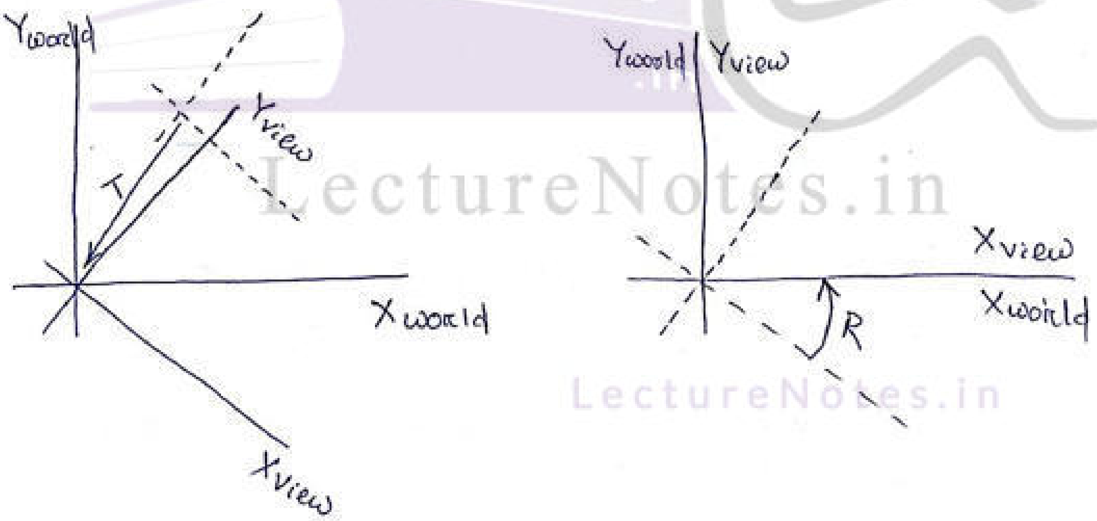
[2D viewing Transformation Pipeline]

Viewing Coordinate Reference Frame

- This coordinate system provides the reference frame for specifying the world coordinate window
- We set up the viewing coordinate system using the transformations between coordinate systems.
- First a viewing-coordinate origin is selected at some world position : $P_0 = (x_0, y_0)$
- To obtain the matrix for converting world-coordinate positions to viewing coordinates we do the following:
  1) Translate the viewing origin to the world origin
  2) Rotate to allign the two coordinate reference frames

Mathematically $M_{WC,VC} = R \cdot T$

~~Window-to-Viewport Coordinate Transformation~~



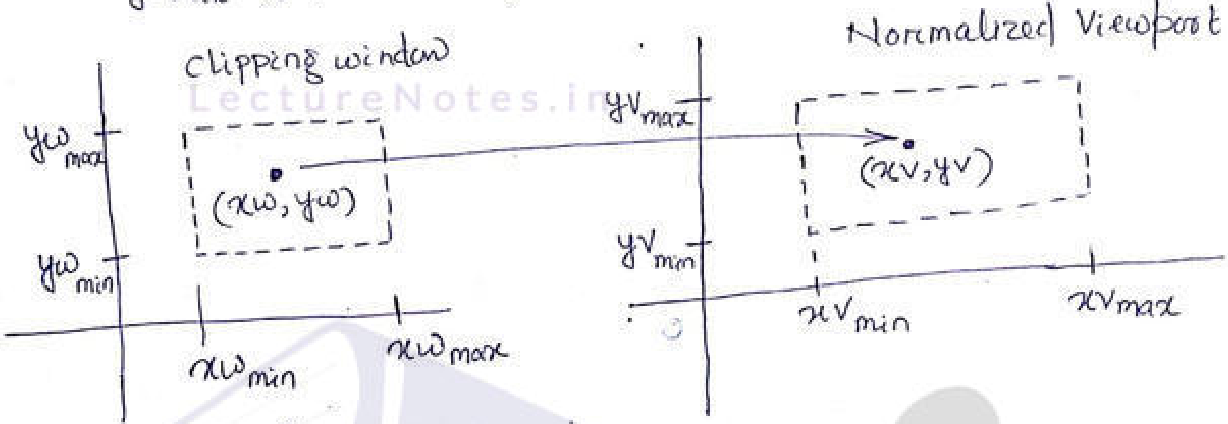
Window-to-Viewport Coordinate Transformation

Consider a point (x_w, y_w) inside the clipping window.
Let this point has to map to a position (x_v, y_v) inside the viewport.

To maintain the same relative placement in the viewport as in the window, we need

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

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



Solving for (xv, yv) we have:

$xv = s_x xw + t_x$

$yv = s_y yw + t_y$ where

Scaling factors:

$$s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}, \quad s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

Translation factors:

$$t_x = \frac{xw_{max}\, xv_{min} - xw_{min}\, xv_{max}}{xw_{max} - xw_{min}}, \quad t_y = \frac{yw_{max}\, yv_{min} - yw_{min}\, yv_{max}}{yw_{max} - yw_{min}}$$

This can also be obtained by composing transformations:

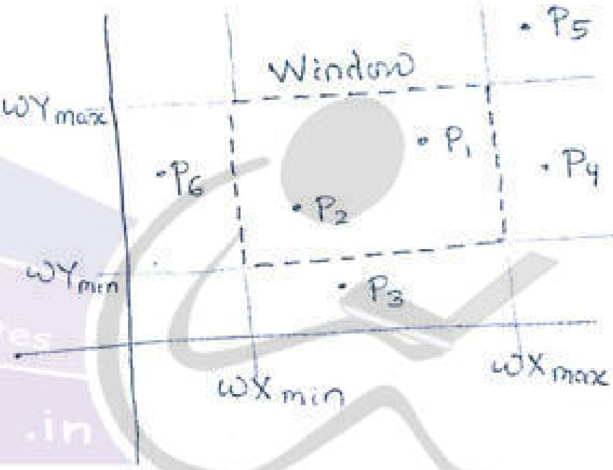
$M_{window,\ norm\_viewport}$

$= T(xv_{min}, yv_{min}) \cdot S(s_x, s_y) \cdot T(-xw_{min}, -yw_{min})$

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

Clipping

- Any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a clipping algorithm or clipping.
- The region against which an object is to be clipped is called a clip window.
- Types of Clipping:
  a) Point Clipping
  b) Line Clipping (straight-line segments)
  c) Area Clipping (polygons)
  d) Curve clipping
  e) Text clipping.



Point Clipping

A point $P(x, y)$ is not clipped and visible if

$$wx_{min} \leq x \leq wx_{max}$$

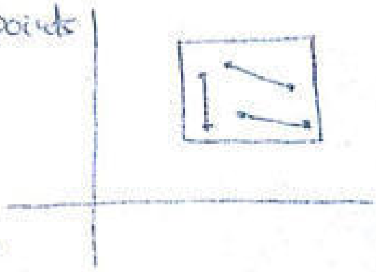
and $wy_{min} \leq y \leq wy_{max}$

otherwise it is clipped.

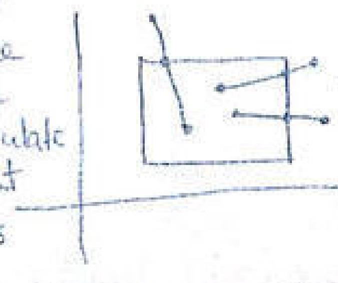
Eg: The points P_3, P_4, P_5, P_6 will be clipped

Line Clipping

- Examine the end points of each line to see if they are in the window or not
- The Brute Force line clipping can be performed as follows:
  - ✓ Don't clip lines with both end-points within the window

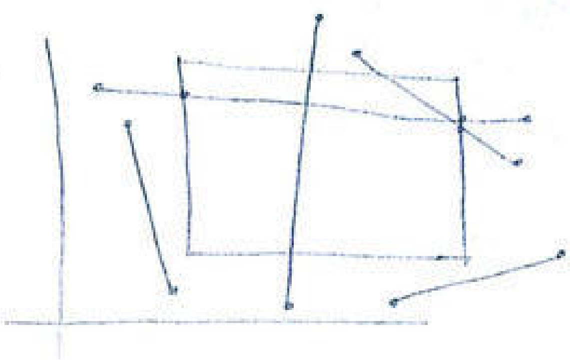


  - ✓ For lines with one end-point inside the window and one end point outside, calculate the intersection point and clip from this point out.



✓ For lines with both end points outside the window, test the line for intersection with all of the window boundaries and clip appropriately.

However, calculating line intersections is computationally expensive.



Cohen-Sutherland Line Clipping Algorithm

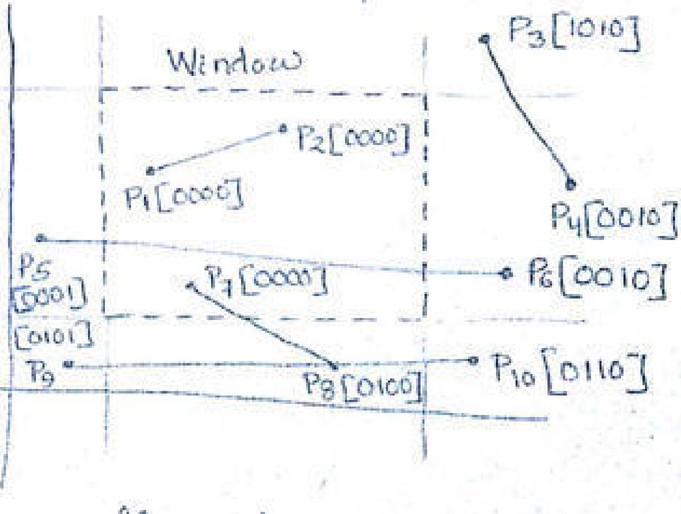
- This is the efficient, oldest and popular line-clipping algorithm.
- Advantage: It vastly reduces the number of line intersections that must be calculated
- World space is divided into regions based on the window boundaries
  - ✓ Each region has a unique four bit region code
  - ✓ Region codes indicate the position of the regions with respect to the window

3	2	1	0
above	below	right	left

Region Code Legend

1001	1000	1010
0001	0000 Window	0010
0101	0100	0110

- Every end-point is labelled with the appropriate region code
- Lines completely contained within the window boundaries have region code [0000] for both end points so are not clipped.
- Any lines with a common set bit in the region codes of both end points can be clipped
  - ✓ The AND operation can efficiently check this.



Algorithm

Step 1 : Compute the 4 bit code for each region

i) Above = 1, if $Y > Y_{max}$
0, otherwise

ii) Below = 1, if $Y < Y_{min}$
0, otherwise

iii) Right = 1, if $X > X_{max}$
0, otherwise

iv) Left = 1, if $X < X_{min}$
0, otherwise

Step 2 : Compute the 4 bit code for the two end points of the line

- ✓ If the two end points are 0000, the line is inside the clipping window & no clipping is required

Step 3 : If one end point code is 0000 and other is not 0000, then the line must be clipped

Step 4 : If both end points are not 0000, then compute the logical AND operation between both bit codes

- ✓ If the result is not 0000, line is totally outside the window & is trivially rejected.
- ✓ Otherwise we will go for clipping.

Step 5 : Let the line has the end points (x_1, y_1) & (x_2, y_2)

- ✓ The y-coordinate of an intersection with a vertical window boundary can be calculated using:

Point : (x_{min}, y) $\quad y = y_1 + m(x_{min} - x_1)$ Left edge intersection

Point : (x_{max}, y) $\quad y = y_1 + m(x_{max} - x_1)$ Right edge intersection

- ✓ The x-coordinate of an intersection with a horizontal window boundary can be calculated using:

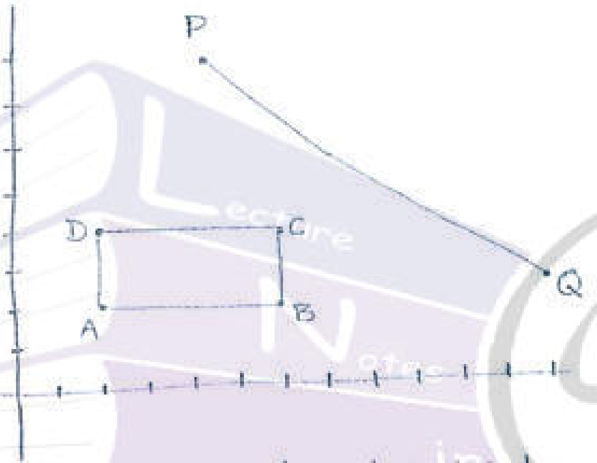
Point : (x, y_{max}) $\quad x = x_1 + (y_{max} - y_1)/m$ Top edge intersection

Point : (x, y_{min}) $\quad x = x_1 + (y_{min} - y_1)/m$ Bottom edge intersection

where $m = (y_2 - y_1)/(x_2 - x_1)$

Step 6
- ✓ After finding the intersection points, compare with the point clipping algorithm.
- ✓ Then choose which pair of intersection point satisfies the point clipping algorithm.
- ✓ This pair of intersection point is visible inside the clipping window.

Example 1 Given a clipping window with co-ordinates A(20,20), B(60,20), C(60,40), D(20,40) using COHEN-SUTHERLAND line-clipping algorithm. Find the visible of the line joining the points P(40,80), Q(120,30)



$X_{min} = 20$
$Y_{min} = 20$
$X_{max} = 60$
$Y_{max} = 40$

Point	4 Bit Code
P	1000
Q	0010
	0000 (Logical AND)

As the bit code of the end points P, Q are not 0000 so clipping is needed.

Result is 0000. So we go for clipping.

Intersection Points

slope of the line, $m = \frac{30-80}{120-40} = \frac{-50}{80} = -5/8$

$x_1 = 40$, $y_1 = 80$

1. Left-edge

$y = y_1 + m(x_{min} - x_1) = -\frac{5}{8}(20-40) + 80 = 92.5 > y_{max}$

So $(x_{min}, y) = (20, 92.5)$ is rejected.

2. Right-edge

$y = y_1 + m(x_{max} - x_1) = -\frac{5}{8}(60-40) + 80 = 67.5 > y_{max}$

So $(x_{max}, y) = (60, 67.5)$ is rejected.

3. Top Edge

$x = x_1 + \frac{1}{m}(y_{max} - y_1) = 40 - \frac{8}{5}(40-80) = 104 > X_{max}$

So $(x, Y_{max}) = (104, 40)$ is rejected.

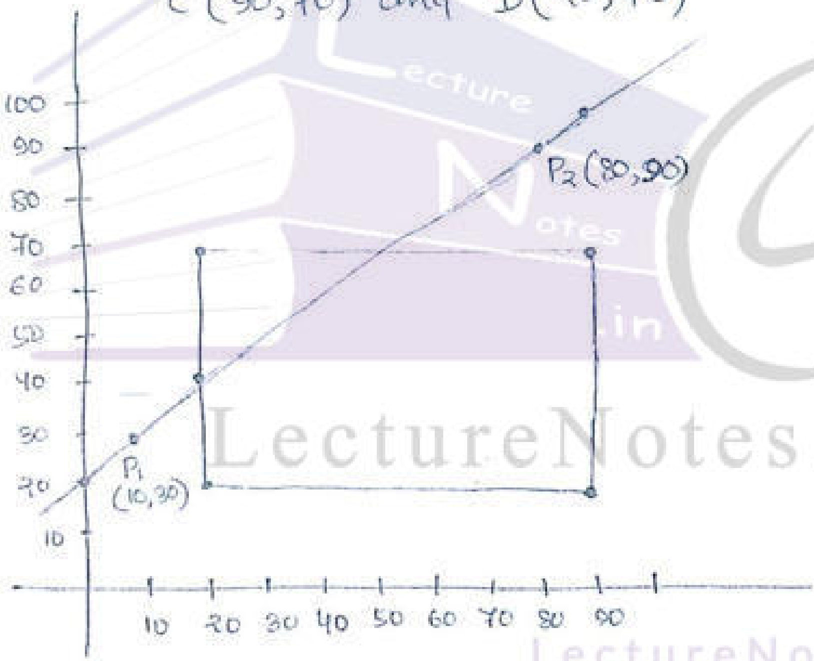
4. Bottom Edge

$x = x_1 + \frac{1}{m}(Y_{min} - y_1) = 40 - \frac{8}{5}(20-80) = 136 > X_{max}$

So $(x, Y_{min}) = (136, 20)$ is rejected.

So no intersection point satisfy the point clipping algorithm. Hence the line is not visible inside the clipping window.

Example 2 Apply the Cohen-Sutherland line-clipping algorithm to clip the line P_1P_2 with $P_1(10,30)$ and $P_2(80,90)$ against the window ABCD with A(20,20), B(90,20) C(90,70) and D(20,70)



$X_{min} = 20$
$Y_{min} = 20$
$X_{max} = 90$
$Y_{max} = 70$

Point	4 Bit Code
P_1	0001
P_2	1000
	0000 (Logical AND)

As the bit code of the end points P_1, P_2 are not 0000. So clipping is needed.

Result is 0000. So we go for clipping.

Intersection Points

slope of the line, $m = \frac{90-30}{80-10} = \frac{6}{7}$

$x_1 = 10$, $y_1 = 30$

1. Left-edge

$y = y_1 + m(x_{min} - x_1) = 30 + \frac{6}{7}(20-10) = 38.571 < y_{max}$

So $(x_{min}, y) = (20, 38.571)$ is selected.

2. Right-edge

$y = y_1 + m(x_{max} - x_1) = 30 + \frac{6}{7}(90-10) = 98.571 > y_{max}$

So $(x_{max}, y) = (90, 98.571)$ is rejected.

3. Top-edge

$x = x_1 + \frac{1}{m}(y_{max} - y_1) = 10 + \frac{7}{6}(70-30) = 56.66 < x_{max}$

So $(x, Y_{max}) = (56.66, 70)$ is selected.

4. Bottom-edge

$x = x_1 + \frac{1}{m}(y_{min} - y_1) = 10 + \frac{7}{6}(20-30) = -1.666 < x_{min}$

So $(x, Y_{min}) = (-1.666, 20)$ is rejected.

Polygon Clipping

Sutherland - Hodgeman Polygon clipping

- This algorithm is suitable for convex type polygon.

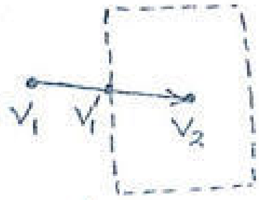
Convex Polygon [Joining any two points will be inside the polygon]

Concave Polygon

- For clipping a polygon inside a rectangular view-port it needs to be checked for all four window boundaries but during the construction of polygon either it is drawn clockwise or anticlockwise direction

- At each step a new sequence of vertices are generated and passed to the next window boundary clipper. Possible cases for the vertices to be clipped are as follows:

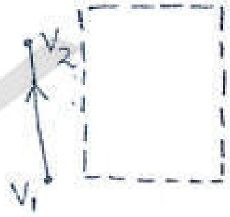
Case I : If the first vertex (V_1) is outside the window boundary and the second vertex (V_2) is inside, both intersection point of the polygon edge (V_1') with the window boundary and the second vertex (V_2) are added to the output vertex list.



Case II : If both input vertices (V_1, V_2) are inside the window boundary, only the second vertex (V_2) is added to the output vertex list.

Case III : If the first vertex (V_1) is inside the window boundary and the second vertex (V_2) is outside, only the edge intersection with the window boundary (V_1') is added to the output vertex list.

Case IV : If both input vertices (V_1, V_2) are outside the window boundary, nothing is added to the output list.



Algorithm

Step 1 : Read the coordinates of vertices of the subject polygon and clipping polygon (window)

Step 2 : Consider an edge of the clipping window and compare the vertices of each edge of the subject polygon with the clipping window plane on the edge and record the intersection.

Step 3 : Store the new intersection and output vertices in the new list of vertices as per the cases discussed above.

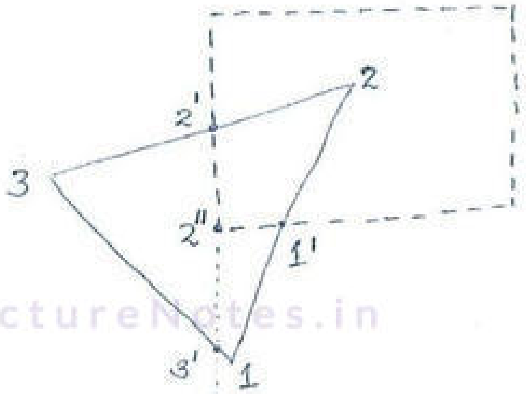
Step 4 : Perform Step 2 and 3 for the remaining edges of the clipping polygon each time, the resulting list of polygon vertices are successively passed to process the next edge of the clipping polygon.

Step 5 : Finish.

Drawback

- It is not suitable for concave type polygon.

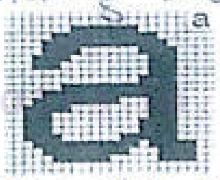
Example:



Left clip	Right clip	Bottom clip	Top clip
(1,2) : (I/I) → {2}	(2,2') : (I/I) → {2'}	(2',3') : (I/O) → {2''}	(2'',1') : (I/I) → {1'}
(2,3) : (I/O) → {2'}	(2',3') : (I/I) → {3'}	(3',1) : (O/O) → { }	(1',2) : (I/I) → {2}
(3,1) : (O/I) → {3',1}	(3',1) : (I/I) → {1}	(1,2) : (O/I) → {1',2}	(2,2') : (I/I) → {2'}
	(1,2) : (I/I) → {2}	(2,2') : (I/I) → {2'}	(2',2'') : (I/I) → {2''}

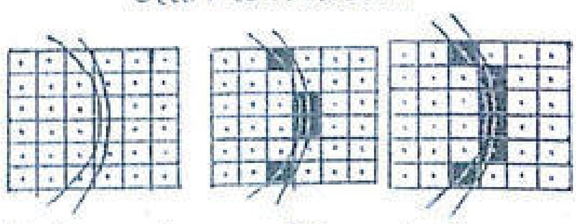
Scan Conversion of Characters

- To display the text of a document we need to map characters to abstract glyphs.
- A glyph is the actual artistic representation in some typographical style in the form of outlines or bitmaps that may be drawn on the screen or paper.
- Character fonts on raster-scanned display devices are usually represented by arrays of bits that are displayed as a matrix of black and white dots.
- Displaying characters on the screen generally sets the frame buffer to background intensity or color.
- A mask is used to write characters in the same buffer. A mask is a boolean array, defining the character storing 0 or 1 for a black and white display. For a gray scale and multicolor character display multiple additional bits are used.
- There are three basic kinds of computer font file data formats:
  a) Bitmap fonts consists of a series of dots or pixels representing the image of each glyph.
  b) Outline fonts use Bezier curves, drawing instructions and mathematical formulas to describe each glyph.
  c) stroke fonts use a series of specified lines and additional information to describe the appearance of the glyph.
- A scan conversion is the job of coloring inside the character outlines contained in the font.



A Character after scan conversion

- Scan converter is able to maintain the continuity of character bitmaps by performing dropout control.
- Dropout occurs when the space within the outlines becomes so narrow that pixel centers are missed



A character outline before and after a dropout control

- The process of a scan conversion consists of four steps:
  1. Measurement: The outline of the character is traversed point by point and contour (outline) by contour in order to find the maximum and minimum coordinate values of the outline. The amount of workspace memory that will be needed to perform next two steps is also calculated in this step.
  2. Rendering: Every contour is broken into lines and splines. Calculations are made to find the point at which each line or spline intersects with scan lines (lines passing through bitmap pixel centers). These intersections are sorted from left to right.
  3. Filling: Using the sorted intersections, runs of pixels are set for each scan line of the bit map from top to bottom.
  4. Dropout control: If dropout control is enabled, the intersection list is checked again looking for dropouts. The dropout control requires scanning in the vertical as well.

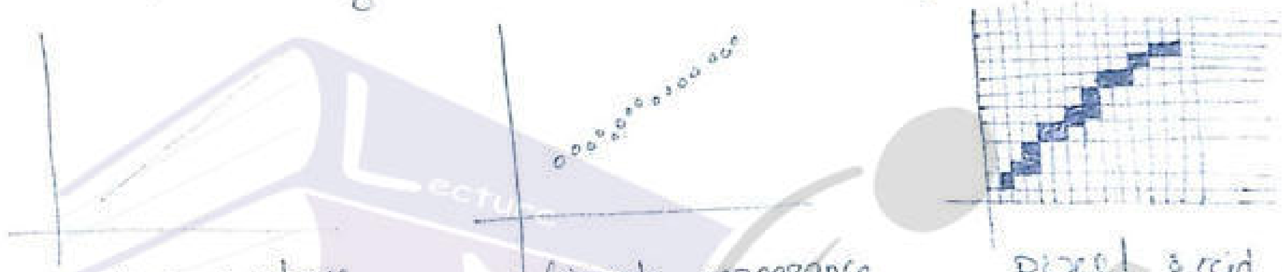
- Bitmap fonts are faster and easier to use in computer code but inflexible, requiring a separate font each size & face.
- Outline and stroke fonts can be resized using a single font and substituting different measurements for components of each glyph, but are more complicated to use.
- A bitmap image can be displayed in a different size only with some distortion but renders quickly. Outline and stroke image formats are resizable but take more time to render.
- Outline fonts can be scaled up to very large size without much quality loss and take less memory space on large sizes. An unscaled bitmap font gives the best possible letter quality, but scaling a bitmap font results in rapid degradation in quality.

Aliasing

- Scan conversion is essentially a systematic approach for mapping objects that are defined in continuous space (world coordinates) to their discrete approximation (pixel coordinates)

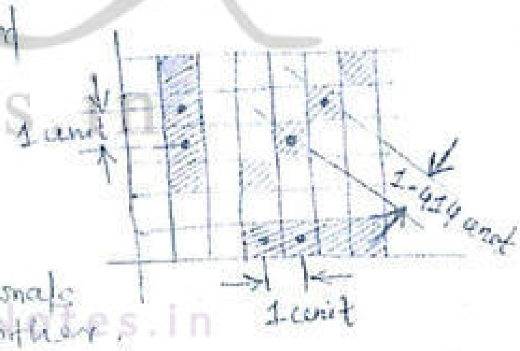
- Scan conversion introduces various forms of distortions collectively known as <u>aliasing</u> effect of a scan conversion
- Some examples of aliasing effects are:

1. Staircase appearance: The screen is made up of pixels in a grid formation, due to which it is impossible to avoid giving most discrete lines and circles a staircase effect (the jaggies). The monitor are capable of producing nearly perfect straight lines either horizontally or vertically. However, when it comes to diagonal lines of any angle, a computer monitor is not capable of producing a line without some jagged edge.

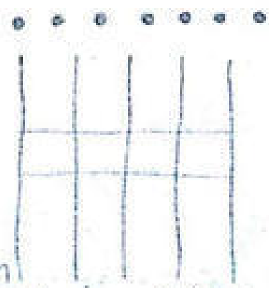


continuous space (world coordinate) — discrete appearance (device coordinate) — pixel grid in device coordinate

2. Unequal brightness: Horizontal and vertical lines appear much brighter than slanted ones although both are painted with same intensity.
Reason - Horizontal and vertical pixels are one unit apart while the pixels on the diagonals are 1.414 units from each other.



3. Picket - fence problem: This problem occurs when an object does not fit into the pixel grid properly.
The figure shows the distance between two adjacent pickets is not a multiple of the unit distance between the pixels. This results



- ✓ Global aliasing: Scan conversion into image space with uneven distance between pickets, since the end points of the pickets will have to be snapped to pixel coordinates.
- ✓ Local aliasing: To maintain equal spacing between pickets we have to distort the overall <u>length of the fence.</u>

Solutions for aliasing effects

There are two methods to deal with aliasing effects

1. Increasing image resolution:

   By increasing image resolution the effect of aliasing can be reduced. But we pay a heavy price in terms of system resource (more memory) and the results are not always satisfactory.

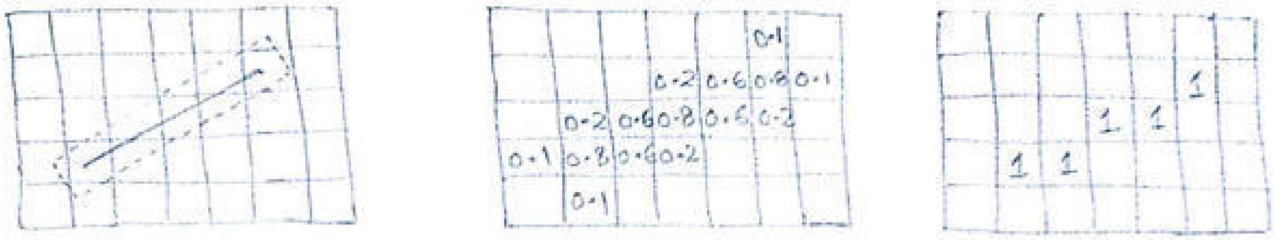
2. Anti-aliasing:

   - It is a technique of representing a high-resolution image (real image) at a lower resolution (on monitor) to minimize the aliasing effect.
   - It is a method of fooling the eye that a jagged edge is really smooth.
   - It is often used in games, digital photography and on graphics cards.
   - We can apply anti-aliasing methods to modify pixel intensities along the boundaries of primitives (straight line, curve, conic etc.), we can smooth the edges to lessen the jagged appearance.
   - It needs a large amount of computation time.

Types of Anti-aliasing

a.) Area Sampling

- It is a pre-filtering technique in which we superimpose a pixel grid pattern onto the continuous object definition.
- For each pixel area that intersects the object, we calculate the percentage of overlap by the object.
- The higher the percentage of overlap, the greater influence the object has on the pixel's overall intensity value.
- The following figure(a) shows a line (solid line) which is represented by a rectangular region (dotted line) of one pixel wide.

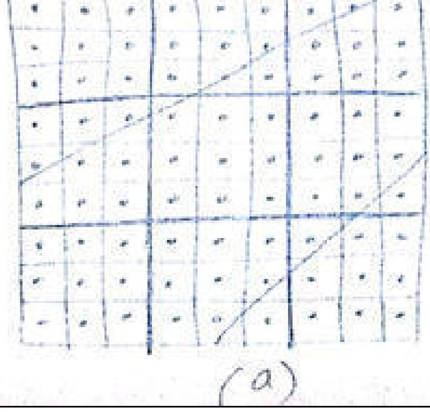


(a) (b) (c)

- Figure (b) shows the percentage of overlap between the rectangle and each intersecting pixel which is calculated analytically.
- If the background is black and the line is white the percentage value is used directly to set the intensity of pixels (shown in fig (b)).
- If the background is gray (0.5, 0.5, 0.5) and the line green (0, 1, 0), each blank pixel in the grid will have background gray value and each pixel filled with a fractional number f will have a colour whose values are (0.5(1-f), 0.5(1-f)+f, 0.5(1-f))
- Figure (c) shows the values of an ordinary scan-conversion method.

b) <u>Super Sampling</u>

- We subdivide each pixel into subpixels and check the position of each subpixel in relation to the object to be scan converted.
- The objects contribution to a pixel's overall intensity value is proportional to the number of subpixels that are inside the area occupied by the object.



0	2/9	7/9
6/9	1	8/9
1	8/9	1/9

(a) (b)

83

- The figure shows an example where we have a white object that is bounded by two slanted lines on a black background.
- Each pixel is subdivided into 9 (3×3) subpixels.
- The pixel values is shown in the figure (b)
- The upper right pixel is assigned to 7/9. If the object is red in colour with values (1,0,0) and the background is light yellow with values (0.5, 0.5, 0) the pixel will be assigned $\left(1\times\frac{7}{9}+0.5\times\frac{2}{9},\ 0.5\times\frac{2}{9},\ 0\right)=\left(\frac{8}{9},\frac{1}{9},0\right)$
- It is a post-filtering technique since discrete samples are first taken and then used to calculate pixel values.
- It is an approximation to area sampling method.

c) Pixel Phasing

- It is a hardware-based anti-aliasing technique
- The graphical system is capable of shifting individual pixels from their normal positions in the pixel grid by a fraction (1/4 and 1/2) of the unit distance between pixel
- This technique is very effective in smoothing out the stair_steps without reducing the sharpness of the edges.

NOTE Pre-filtering and Post-filtering

- These are two types of general-purpose anti-aliasing techniques.
- Filtering concept is originated from the field of signal processing.
- A pre-filtering technique works on the true signal in the continuous space to derive proper values for individual pixels (filtering before sampling), whereas a post-filtering technique takes discrete samples of the continuous signal and uses the samples to compute pixel values (sampling before filtering)

Halftoning

- Continuous-tone photographs (real images, digital photographs) has infinite range of intensities, however an output device (eg: printer) has a limited intensity[1] range.
- How can we expand the range of available intensities?
  If we view a very small area from a sufficiently large viewing distance, our eyes average fine detail within the small area and record only the overall intensity of the area.
- Continuous-tone photographs are reproduced for publication in newspapers, magazines and books with a printing process called halftoning and the produced pictures are called halftones.
- Halftone images contain a series of dots in a specific patterns that simulate the look of a continuous-tone image
- For a black-and-white photograph, each intensity area is reproduced as a series of black circles on a white background. The diameter of each circle is proportional to the darkness required for that intensity region. Darker regions are printed with large circles and lighter regions with smaller circles.
- Color halftones are printed using dots of various size and colour
- Book and magazine halftones are printed on high-quality paper using approximately 60-80 circles of varying diameter per centimeter
- Newspapers use lower-quality paper and lower resolution (approximately 25 to 30 dots per centimeter)

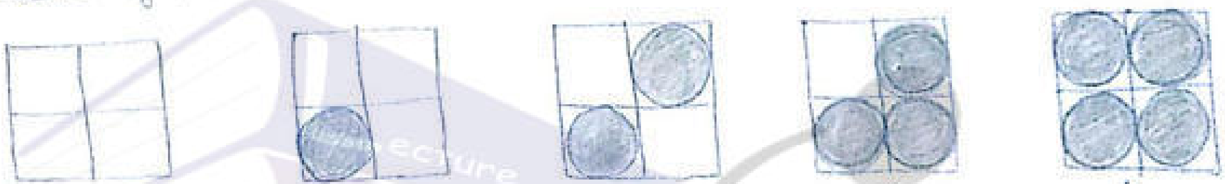
White->Gray->Dark

size of black dot increases on white background

[1] Intensity refers to the amount of light or the numerical value of a pixel. For example in an 8-bit gray scale image there are 256 gray levels. So any pixel can have a value in the range 0 to 255.

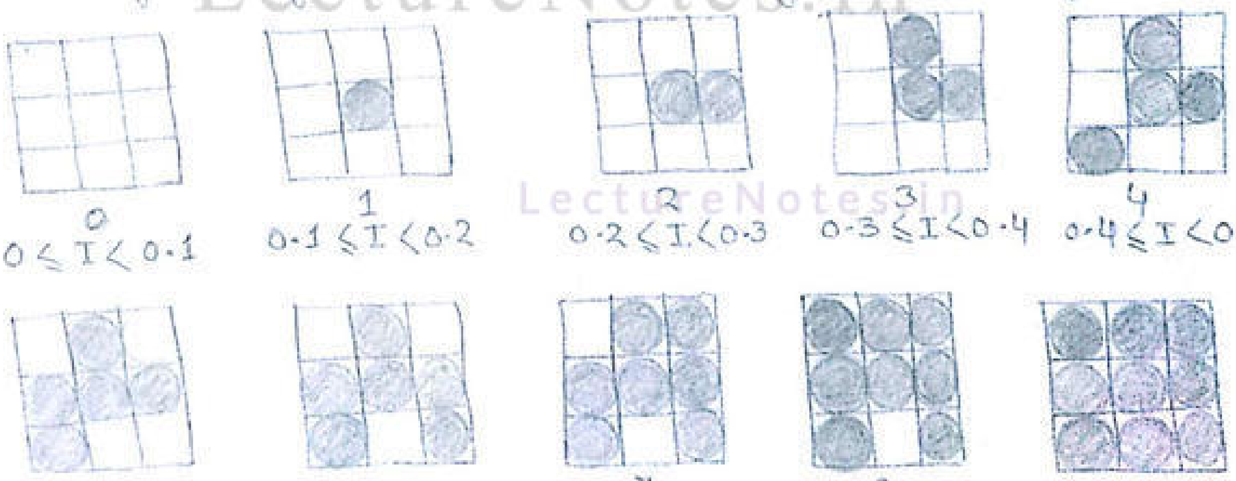
Halftone Approximations

- In Computer Graphics, halftone reproductions are approximated using rectangular pixel regions, called halftone patterns or pixel patterns.
- The number of intensity levels that we can display with this method depends on how many pixels we include in the rectangular grid and how many level the system can display.
- With n by n pixels for each grid on a bilevel system, we can represent n^2+1 intensity levels.
- The following figure shows one way to set up pixel patterns to represent five intensity levels that could be used with a bilevel system.



0	1	2	3	4
$0 \leq I < 0.2$	$0.2 \leq I < 0.4$	$0.4 \leq I < 0.6$	$0.6 \leq I < 0.8$	$0.8 \leq I \leq 1.0$
All pixels are turned off	One pixel is turned on	Two pixels are turned on	Three pixels are turned on	All pixels are turned on

- The following figure shows 3 by 3 pixel grids on a bilevel system by which we can display 10 intensity levels.



0	1	2	3	4
$0 \leq I < 0.1$	$0.1 \leq I < 0.2$	$0.2 \leq I < 0.3$	$0.3 \leq I < 0.4$	$0.4 \leq I < 0.5$

5	6	7	8	9
$0.5 \leq I < 0.6$	$0.6 \leq I < 0.7$	$0.7 \leq I < 0.8$	$0.8 \leq I < 0.9$	$0.9 \leq I \leq 1.0$

- In the grid the "on" pixel positions are near the center of the grid for lower intensity levels and expand outward as the intensity level increases.

- For any pixel-grid size, we can represent the pixel patterns for the various possible intensities with a "mask" of pixel position numbers

Eg: Mask for 2 by 2 grid patterns shown in the previous page is

$$\begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}$$

Similarly the mask for 3 by 3 grid patterns shown before is

$$\begin{bmatrix} 8 & 3 & 7 \\ 5 & 1 & 2 \\ 4 & 9 & 6 \end{bmatrix}$$

To display a particular intensity with <u>level number k</u>, we turn on each pixel whose position number is <u>less than or equal to k.</u>

Disadvantages

- Although the use of n by n pixel patterns increases the number of intensities that can be displayed, they <u>reduce the resolution (sharpness)</u> by a factor of 1/n along each of the x and y axes.

Eg: A 512 X 512 screen area is reduced to an area containing 256 X 256 intensity points by 2 by 2 grid patterns

Similarly the same screen area can be reduced to an area containing 128 X 128 intensity points by 3 by 3 grid patterns.

Thresholding

- It is a technique for improving the visual resolution while maintaining the spatial resolution (no of pixels utilized in construction) of an image.
- Generally, the thresholding technique deals with the problem where we have a digital image with the same resolution as our monochrome display device but with more intensity levels.
- It is the simplest method of image segmentation

- From a gray scale image, thresholding can be used to create binary images (black & white)
- The simplest form of thresholding is to use a fixed threshold for each pixel. If the intensity exceeds that value, the pixel is drawn white, otherwise it is drawn black.

If $I(x,y) > T$ then
    Paint pixel with white colour
Else
    Paint pixel with black colour

$I(x,y)$: Intensity of the image at spatial coordinates (x,y)

T : Threshold value.

- The key parameter in the thresholding process is the choice of threshold value (T).

Dithering

- Dithering is used to create the illusion of "color depth" in images with a limited color palette.
- This technique is also known as color quantization.
- In a dithered image colors that are not available in the palette are approximated by a diffusion of colored pixels from within the available palette.
- It is a color reproduction technique in which dots or pixels are arranged in such a way that allows us to perceive more colors than are actually used.
- It can be used to approximate halftones without reducing the resolution.
- Classical algorithms for dithering include:
  - ✓ average dithering
  - ✓ Ordered dithering
  - ✓ Floyd - Steinberg dithering
  - ✓ Random dithering

Computer Graphics

Topic:

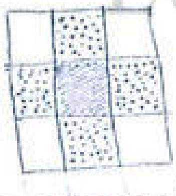
Polygon Filling

Contributed By:

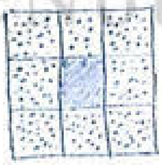
Jasaswi Prasad Mohanty

Polygon Filling

- A chain of connected line segments often called polygons
- Polygons are closed polylines whose starting and ending vertices are same.
- Polygons may be either convex or concave.
- Filling polygon
  - ✓ which pixel to fill
  - ✓ what to fill them
- Fill algorithms deal with pixel-defined regions.
- A pixel-defined region is a group of pixels with the same colours that are connected to one another.
- Two pixels are <u>connected</u> when there is an unbroken path of adjacent pixels connecting them.
- Two common definitions of adjacent pixels:
  - ✓ 4-adjacent : Two pixels are 4-adjacent, if they lie next to each others horizontally or vertically but not diagonally.
  - ✓ 8-adjacent : Two pixels are 8-adjacent, if they lie next to each others horizontally, vertically, or diagonally.



4-connected region



8-connected region

- Techniques of Polygon Filling
  - ✓ Seed-fill Technique :
    - * This technique works by asuming a point called <u>seed point</u> known to be inside the polygon and searches adjacent points near it that are inside the polygon.
    - * If a newly discovered adjacent point is found inside the polygon then it becomes the new seed. and the algorithm continues.

* If the adjacent point found is not to be inside the polygon then the boundary has been found.
* This algorithm only applicable to raster devices.

✓ Scan conversion / Scan-line fill technique
* This techniques are used to determine whether or not a point is inside a polygon using scan-line order.
* The algorithm proceeds from top to bottom of the polygon.
* The algorithm is applicable to raster as well as line drawing displays.

Seed-fill Algorithm

a) Boundary-fill algorithm:

- It is used to fill an area with a specified color until the specified boundary color is encountered.
- The algorithm starts from a specified point (x,y), fills that point with a specified fill color if it is not a boundary and recursively continue with four or eight closest neighbors.

Algorithm

1. Suppose that the edges of the polygon has already been colored.
2. Suppose that the interior of the polygon is to be colored a different color from the edge.
3. Suppose we start a pixel inside the polygon, then we color that pixel and all surrounding pixels until we meet a pixel that is already colored.
4. Start at a point inside the region.
5. Paint the interior outward to the edge
6. The edge must be specified in a single color.
7. Fill the 4-connected or 8-connected region

4-connected Boundary Fill

```
void BoundaryFill4 (int x, int y, int newcolor, int edgecolor)
{
    int current = getPixel(x, y);
    if (current != edgecolor && current != newcolor)
    {
        putPixel (x, y, newcolor);
        BoundaryFill4(x+1, y, newcolor, edgecolor);
        BoundaryFill4 (x-1, y, newcolor, edgecolor);
        BoundaryFill4 (x, y+1, newcolor, edgecolor);
        BoundaryFill4 (x, y-1, newcolor, edgecolor);
    }
}
```

8-connected Boundary Fill

```
void BoundaryFill8 ( int x, int y, int newcolor, int edgecolor)
{
    int current = getPixel (x, y);
    if (current != edgecolor && current != newcolor)
    {
        putPixel (x, y, newcolor);
        BoundaryFill8 (x-1, y, newcolor, edgecolor);
        BoundaryFill8 (x+1, y, newcolor, edgecolor);
        BoundaryFill8 (x, y+1, newcolor, edgecolor);
        BoundaryFill8 (x, y-1, newcolor, edgecolor);
        BoundaryFill8 (x-1, y-1, newcolor, edgecolor);
        BoundaryFill8 (x+1, y+1, newcolor, edgecolor);
        BoundaryFill8 (x-1, y+1, newcolor, edgecolor);
        BoundaryFill8 (x+1, y-1, newcolor, edgecolor);
    }
}
```

Flood Fill Algorithm

1. Suppose we want to color the entire area whose original color is interiorColor and replace it with fillColor.

2. then, we start with a point in this area, then color all surrounding points until we see a pixel that is not interior color.
3. Start at a point inside a region.
4. Replace a specified interior color (old color) with fill color.
5. Fill the 4-connected or 8-connected region until all interior points being placed.

NOTE: This algorithm is used when an area defined with multiple color boundaries.

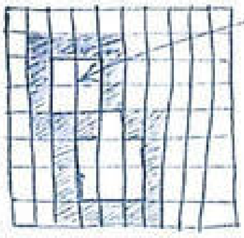
4-connected Flood Fill

```
void FloodFill4(int x, int y, int newcolor, int oldcolor)
{
    if ( getPixel(x,y) == oldcolor)
    {
        putPixel (x, y, newcolor);
        FloodFill4 (x+1, y, newcolor, oldcolor);
        FloodFill4 (x-1, y, newcolor, oldcolor);
        FloodFill4 (x, y+1, newcolor, oldcolor);
        FloodFill4 (x, y-1, newcolor, oldcolor);
    }
}
```

8-connected Flood Fill

```
void FloodFill8(int x, int y, int newcolor, int oldcolor)
{
    if (getPixel(x,y) == oldcolor)
    {
        putPixel (x,y, newcolor);
        FloodFill8 (x+1, y, newcolor, oldcolor);
        FloodFill8 (x-1, y, newcolor, oldcolor);
        FloodFill8 ( x, y+1, newcolor, oldcolor);
        FloodFill8 ( x, y-1, newcolor, oldcolor);
        FloodFill8 (x-1, y-1, newcolor, oldcolor);
        FloodFill8 (x+1, y+1, newcolor, oldcolor);
        FloodFill8 (x-1, y+1, newcolor, oldcolor);
        FloodFill8 (x+1, y-1, newcolor, oldcolor);
    }
}
```

NOTE



4 connected approach will not fill this area.

There is a problem in 4-connected way of filling polygons. Consider the polygon shown above. When we try to fill the entire region using 4-connected approach it will fill it partially. However if we use 8-connected it can fill it fully.

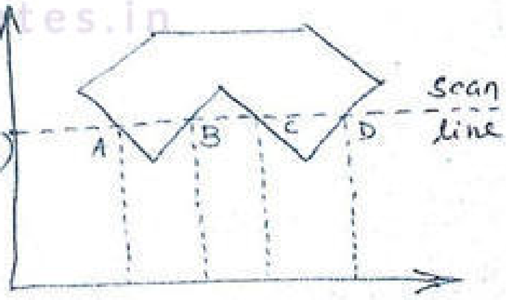
Limitations

1. If any of the inside point or pixel is already in the specified color then recursive call terminates, leaving further pixel unpainted.
2. Not suitable for large polygon fills, because they use stack structures to store the neighbouring pixels that consumes a large memory space.

To avoid these limitations scan-line fill method can be used.

Scan-line algorithm

- It uses the intersection between area boundaries and scan lines to identify pixels that are inside the area.
- The algorithm locates the intersection points of the scan line with each edge of the area to be filled.
- The intersection points are paired (from left to right) and the intervening pixels are set to the specified color.

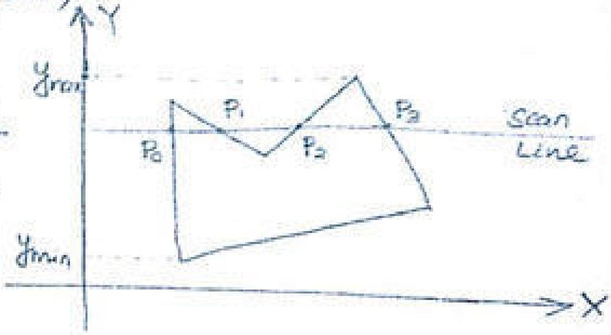


- Eg: A to B and C to D will be colored.

Basic Algorithm

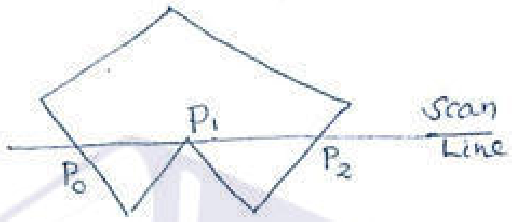
For $y = Y_{min}$ to Y_{max}

1) Intersect scanline y with each edge
2) Sort intersections by increasing x [$P_0, P_1, P_2, P_3, \ldots$]
3) Fill pairwise ($P_0 \to P_1$, $P_2 \to P_3, \ldots$)



Special cases of Handling Scanlines passing through the vertex of polygon

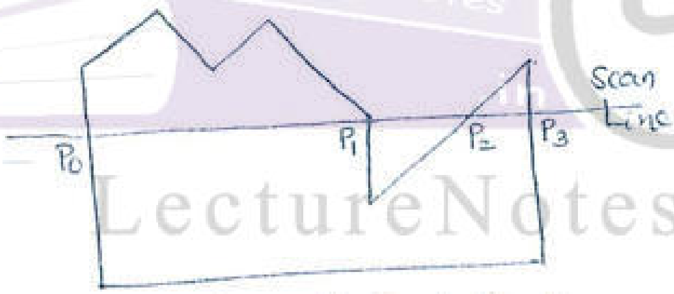
Case I : Edges at the vertex are on the same side of scan line



Intersection points : P_0, P_1, P_2 (odd no of intersection point)

Add one more intersection : P_0, P_1, P_1, P_2 (even no of intersection points)

Case II : Edges at the vertex are on either side of the scan line



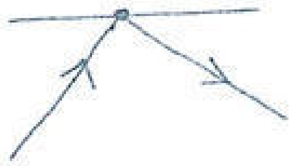
Intersection Points: P_0, P_1, P_2, P_3 (even no of intersection points)

Do not add intersection

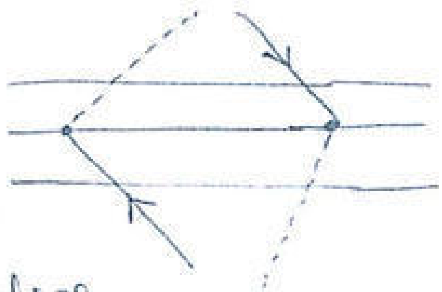
How to know i.e. Case I or Case II ?

- Traverse along the polygon boundary clockwise or anti-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)
- If end point y values of two consecutive edges monotonically increase or decrease, count the middle vertex as a single intersection point for the scanline passing through it.

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



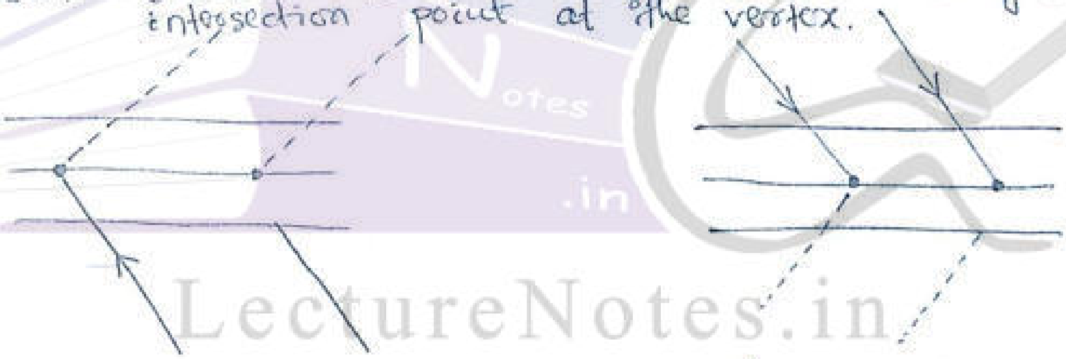
- If the vertex is a __local extrema__, consider (or add) two intersections for the scan line corresponding to such a shared vertex.
- We must avoid this to happen in cases such as:



__How to implement__

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

If so: Shorten the lower edge to ensure only one intersection point at the vertex.



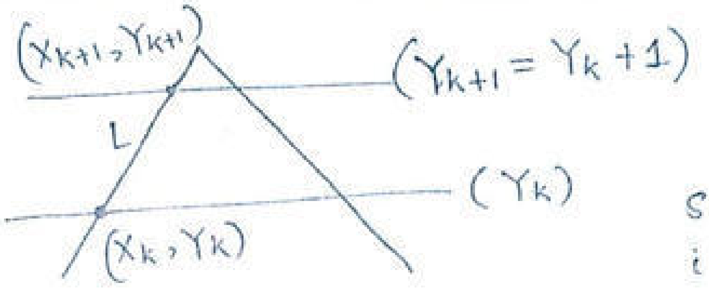
__Important features of scanline-based polygon filling__

1. scanline coherence: value don't change much from one scanline to the next — the visibility or coverage of a face on one scanline typically differs little from the previous one.

2. Edge coherence: Edges intersected by scan line i are typically intersected by scan line i+1

NOTE: Coherence is simply the properties of one part of a scene are related in some way to

Fast calculation of Intersection Points



where $Y_{k+1} = Y_k + 1$

Thus $X_{k+1} = X_k + 1/m$

Slope of the line L (polygon edge) is:

$$m = \frac{Y_{k+1} - Y_k}{X_{k+1} - X_k}$$

Thus the intersection for the next scanline is obtained as:

$x_{k+1} = \text{round}(x_k + 1/m)$ where $m = \frac{\Delta Y}{\Delta X}$

How to implement this using Integer Arithmetic

Take an example $m = \Delta Y / \Delta X = 7/3$

Set Counter, $C = 0$

and counter-increment, $\Delta C = \Delta X = 3$

For the next three scan lines, successive values of C are: 3,6,9

Thus only at 3rd scan line $C > \Delta Y$

Then, X_k is incremented by 1 only at 3rd scanline.

Set C as $C = C - \Delta Y = 9 - 7 = 2$

Repeat the above steps till Y_k reaches Y_{max}

Computer Graphics

Topic:

Two Dimensional Object Representations

Contributed By:

Jasaswi Prasad Mohanty

Representing Curves & Surfaces

- In Computer Graphics, we need to draw different types of objects onto the screen. Objects are not flat all the time and we need to draw curves many times to draw an object.
- A curve is an infinitely large set of points. Each point has two neighbours except end points
- Curves and surfaces can have explicit, implicit and parametric representations.

Implicit Curves

✓ Implicit curve representation define the set of points on a curve by employing a procedure that can test to see if a point is on the curve or not.

✓ Usually, an implicit curve is defined by an implicit function of the form

$$f(x, y) = 0$$

It can represent multivalued curves (multiple y values for an x value)

✓ Example:

Implicit representation of circle is

$$x^2 + y^2 - R^2 = 0$$

Explicit Curves

✓ A mathematical function $y = f(x)$ can be plotted as a curve.

✓ This type of function is the explicit representation of the curve.

✓ Explicit representation is single-valued (for each value of x, only a single value of y is computed).

Parametric Curves

✓ Parametric representations are the most common in Computer Graphics

✓ Curves having parametric form are called parametric curves

- ✓ There are many curves which we can't write down as a single equation in terms of only x and y.
- ✓ Instead of defining y in terms of x $(y = f(x))$ or x in terms of y $(x = h(y))$ we define both x & y in terms of a third variable called a parameter as follows:

$$x = f(t) \qquad y = g(t)$$

Parametric Cubic Curves

- The parametric representation for curves is:

$$x = x(t)$$
$$y = y(t)$$
$$z = z(t)$$

- A curve is approximated by a piecewise polynomial curve instead of piecewise linear curve.
- Each segment Q of the overall curve is given by three functions x, y, z which are cubic polynomials in the parameter "t".
- The cubic polynomials that define a curve segment $Q(t) = [x(t) \quad y(t) \quad z(t)]$ are of the form

$$\left.\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x(t) + d_x \\ y(t) &= a_y t^3 + b_y t^2 + c_y(t) + d_y \\ z(t) &= a_z t^3 + b_z t^2 + c_z(t) + d_z, \quad 0 \le t \le 1 \end{aligned}\right\} \text{——} (1)$$

where $T = [t^3 \quad t^2 \quad t \quad 1]$

- The coefficient matrix is defined as

$$C = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix} \text{——} (2)$$

So we can rewrite equ^n (1) as

$$Q(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix} = T \cdot C \quad \text{——— (3)}$$

The parametric tangent-vector to the curve is

$$\frac{d}{dt} Q(t) = Q'(t) = \left[\frac{d}{dt} x(t) \quad \frac{d}{dt} y(t) \quad \frac{d}{dt} z(t) \right] = \frac{d}{dt} T \cdot C$$

$$= \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \cdot C$$

$$= \left[3a_x t^2 + 2b_x t + c_x \quad 3a_y t^2 + 2b_y t + c_y \quad 3a_z t^2 + 2b_z t + c_z \right] \quad \text{——— (4)}$$

Some Definitions

1. If two curve segments join together, this curve has G^0 geometric continuity. i.e. two successive curve section must have same coordinate position at boundary point.
2. If the directions (but not necessarily the magnitude) of the two segments' tangent vectors are equal at a join point, the curve has G^1 geometric continuity.
   G^1 geometric continuity means that the geometric slopes of the segments are equal at the join point.
   For two tangent vectors TV_1 & TV_2 to have the same direction we should have
   $$TV_1 = k \cdot TV_2 \quad \text{with } k > 0$$
3. If the tangent vectors of two cubic curve segments are equal (i.e. their direction and magnitude are equal) at the segments' join point the curve has first degree continuity in the parameter t or parametric continuity and is said to be C^1 continuous.
4. If the direction & magnitude of $\frac{d^n}{dt^n}[Q(t)]$ through the n-th derivative are equal at the join points, the curve is called C^n continuous.

So we can rewrite (1) as :

$$Q(t) = [x(t) \quad y(t) \quad z(t)] = T.C \quad \text{———} (3)$$

the parametric tangent-vector to the curve is:

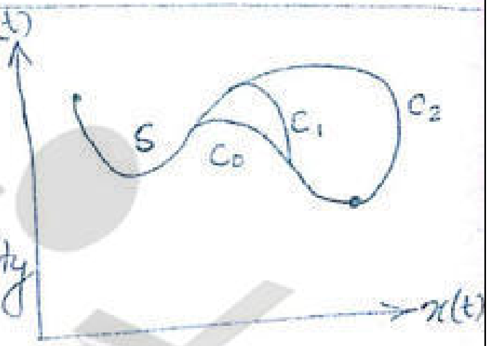
$$\frac{d}{dt} Q(t) = Q'(t) = \left[\frac{d}{dt}.x(t) \quad \frac{d}{dt} y(t) \quad \frac{d}{dt} z(t)\right]$$

$$= \frac{d}{dt} T.C = [3t^2 \; 2t \; 1 \; 0].C$$

$$= [3a_x t^2 + 2b_x t + c_x \quad 3a_y t^2 + 2b_y t + c_y \quad 3a_z t^2 + 2b_z t + c_z]$$

———(4)

The figure shows with three different degrees of continuity. Segment S is joined to segments C_0, C_1, C_2 with the 0, 1, 2 degrees of parametric continuity respectively.



The tangent vector $Q'(t)$ is the velocity of a point on the curve with respect to parameter t. Similarly $Q''(t)$ is the acceleration.

Example: If a camera is moving along a parametric cubic curve in equal time-steps and records a picture after each step, the tangent vector gives the velocity of camera.

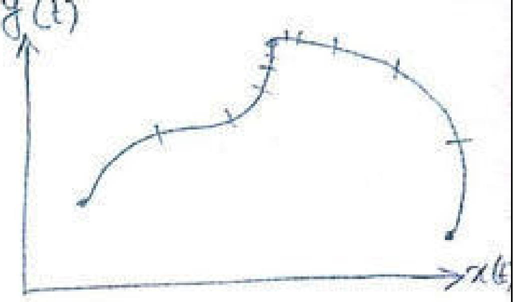
The camera velocity and acceleration at join points should be continuous, to avoid jerky movements in the resulting animation sequence

NOTE:

- In general C^1 continuity implies G^1 but the converse is not true generally.
- Join points with G^1 continuity will appear just as smooth as those with C^1 continuity.

- Special case: C^1 continuity does not imply G^1 continuity when both segments' tangent vectors are $[0\ 0\ 0]$ at the join point.

  In this case the tangent vectors are equal but there directions can be different.



- For geometric-continuity direction is important not the magnitude.

  Example: Let $\gamma(t)$, $0 \leq t \leq 1$ is a parametric curve.

  Tangent of this vector at time 0 is $\gamma'(0)$

  Let $\eta(t) = \gamma(2t)$, $0 \leq t \leq \frac{1}{2}$ be another parametric curve.

  The parametric plots of γ & η are identical

  But $\eta'(0) = 2\gamma'(0)$

  So two curves that have identical plots can have different tangent vectors.

- For smooth joining, the direction of tangent vectors should match at the end-point not their magnitude.

We know already

$$Q(t) = [x(t)\ \ y(t)\ \ z(t)] = T.C$$

We can rewrite the coefficient matrix as

$$C = M.G$$

where M is a 4×4 basis matrix

G is a four-element column vector of geometric constraints called the geometry vector.

The geometric constraints are conditions such as end points or tangent vectors that define the curve.

Now $Q(t) = T.M.G = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix}$ ——— (5)

G_x: column vector of just the x components of the geometry vector.

Similarly G_y & G_z.

Multiplying out just $x(t) = T.M.G$ gives

$$x(t) = (t^3 m_{11} + t^2 m_{21} + t m_{31} + m_{41}) g_{1x} + (t^3 m_{12} + t^2 m_{22} + t m_{32} + m_{42}) g_{2x}$$

$$+ (t^3 m_{13} + t^2 m_{23} + t m_{33} + m_{43}) g_{3x} + (t^3 m_{14} + t^2 m_{24} + t m_{34} + m_{44}) g_{4x}$$ ——— (6)

Similarly we can find $y(t)$ & $z(t)$.

The eq^n(6) emphasizes that the curve is a weighted sum of the elements of the geometry matrix. The weights are each cubic polynomials of t and are called blending functions.

The blending functions, B are given by $B = T.M.$

Curve Design

Spline: A spline is a flexible strip that is used to produce a smooth curve through a set of points. Mathematically, this is defined as a piece-wise cubic polynomial function whose first and second derivative are continuous across the various curve section.

- A spline curve is designed by set of coordinate points known as control points.
- Spline Types:

a) Interpolating Spline Curve: When a curve is passed through all the control points the resulting curve is known as interpolating spline curve.

Ex: Hermite Curve

b) Approximating Spline curve: When a curve is passed through some or all the points; the resulting curve is known as approximating spline curve.

eg: Bezier curve

HERMITE CURVE

- The Hermite form of the cubic polynomial curve segment is determined by constraints on the end points P_1 & P_4 and the tangent vectors at the end points R_1 and R_4.

For Hermite polynomial

$$Q(t) = T \cdot M_H \cdot G_H$$

where M_H : Hermite basis matrix
G_H : Hermite geometry vector.

$$G_{H_x} = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x$$ = The x-component of the geometry matrix

We have

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x = T \cdot C_x = T \cdot M_H \, G_{Hx}$$

$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} M_H \, G_{Hx}$$

$$x(0) = P_{1x} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} M_H G_{Hx}$$ [At point P_1 : $t = 0$

$$x(1) = P_{4x} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} M_H G_{Hx}$$ [At point P_4 : $t = 1$

The tangent-vector constraint equations can be written as:

$$x'(0) = R_{1x} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} M_H \cdot G_{Hx}$$

$$x'(1) = R_{4x} = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} M_H \cdot G_{Hx}$$

So

$$G_{Hx} = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} M_H \cdot G_{Hx}$$

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

the Hermite Blending function is $T.M_H$ i.e/ $B_H = T.M_H$

So $Q(t) = T.M_H.G_H = B_H.G_H$

$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} \quad \text{———— (7)}$$

$$Q(t) = (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4$$

NOTE: Hermite curve has local control over the curve.

BEZIER CURVE

For cubic parametric Bezier form of the curve two end points and two other control points are taken which control the tangent vectors at the end points.

Here P_2, P_3 are the two intermediate points which control the tangent vectors at the end points.

The starting and ending tangent vectors are determined by the vectors P_1P_2 & P_3P_4 and are related to R_1 & R_4 by

$$\left.\begin{aligned} R_1 &= Q'(0) = 3(P_2 - P_1) \\ R_4 &= Q'(1) = 3(P_4 - P_3) \end{aligned}\right\} \quad \text{———— (8)}$$

The Bezier Geometry Vector, $G_B = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$

We can relate the Hermite geometry vector G_H and Bezier geometry vector G_B as

$$G_H = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = M_{HB} \cdot G_B$$

From Hermite Curve we know

$$Q(t) = T \cdot M_H \cdot G_H = T \cdot M_H \cdot M_{HB} \cdot G_B = T \cdot M_B \cdot G_B$$

where $M_B = M_H \cdot M_{HB}$

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

$$= \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Now $Q(t) = T \cdot M_B \cdot G_B$

$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

$$= (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4$$

$$= \sum_{i=0}^{n} P_{i+1} \, B_{i,n}(t)$$

[If control points are P_0, P_1, P_2, P_3 then replace P_{i+1} by P_i in the eqn]

where P_i : set of control points

$B_{i,n}(t)$: Blending function of the Bezier Curve.

$n = 3$ for cubic polynomial.

The blending function is defined using a Benstein basis function as:

$$B_{i,n}(t) = {}^{n}C_{i}\, t^{i} (1-t)^{n-i} \quad \text{where } {}^{n}C_{i} = \frac{n!}{i!(n-i)!}$$

Properties of Bezier Curves

1. They generally follow the shape of the control polygon, which consists of the segments joining the control points.
2. They always pass through the first and last control points.
3. The curve is contained within convex hull of defined polygon
4. The degree of the polynomial defining the curve segment is one less than the number of defining control polygon point. For 4 control points, the degree of the polynomial is 3 i.e. cubic polynomial.
5. The basis function are real.
6. The order of the polynomial defining the curve segment is equal to the total no of control points.
7. No straight line intersects a Bezier curve more than it intersects its control polygon

Drawbacks of Bezier Curve

1. The degree of the Bezier Curve depends on number of control points
2. Bezier curve exhibit global control means moving a control point alters the shape of the whole curve

Problem: Find the equation of the Bezier curve which passes through points (0,0) & (-2,1). The curve is controlled through points (7,5), (2,0).

Solu^n: $$Q(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 0 \\ -2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 13 & 16 \\ -36 & -30 \\ 21 & 15 \\ 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} (13t^3 - 36t^2 + 21t) & (16t^3 - 30t^2 + 15t) \end{bmatrix}$$

Problem Draw the Bezier curve passing through the control points $P_0(40,40)$, $P_1(10,40)$, $P_2(60,60)$ & $P_3(60,0)$ with $t = 0.2, 0.4$ & 0.6

Solu^n: $$Q(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 40 & 40 \\ 10 & 40 \\ 60 & 60 \\ 60 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -130 & -100 \\ 240 & 60 \\ -90 & 0 \\ 40 & 40 \end{bmatrix}$$

$$= \begin{bmatrix} -130t^3 + 240t^2 - 90t + 40 & -100t^3 + 60t^2 + 40 \end{bmatrix}$$

$$Q(0.2) = \begin{bmatrix} 30.56 & 41.6 \end{bmatrix}$$
$$Q(0.4) = \begin{bmatrix} 34.08 & 43.2 \end{bmatrix}$$
$$Q(0.6) = \begin{bmatrix} 44.32 & 40 \end{bmatrix}$$

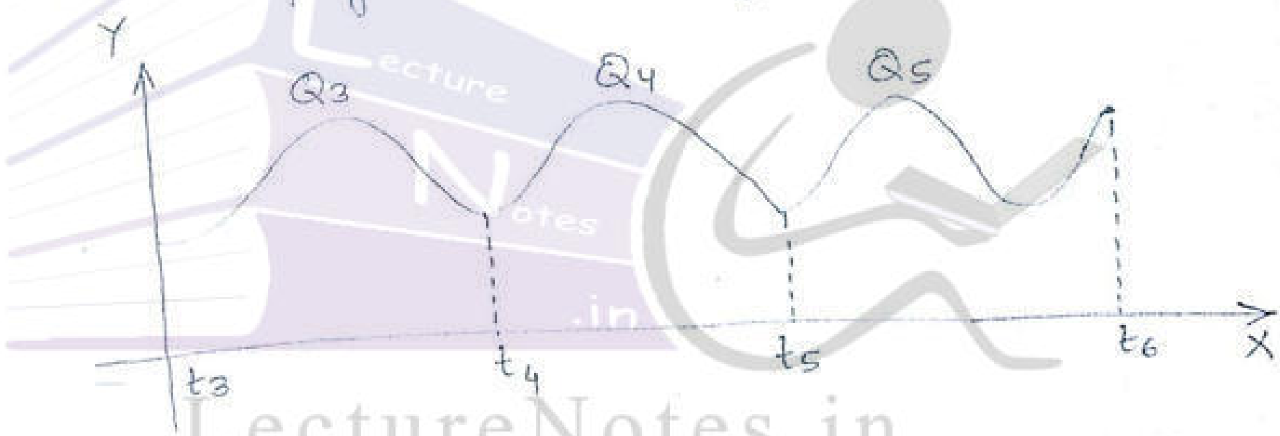
107

B-Spline Curve

Mainly there are two types of B-Spline Curves
- Uniform Nonrational B-Splines
- Non-uniform, Non-rational B-Splines

Uniform Nonrational B-Spline

- B-Spline consists of curve segments whose polynomial coefficients depend on just a few control points. This is called local control because this control point affect only a small part of a curve.
- The curve need not pass through the control points
- Cubic B-Splines approximate a series of m+1 control points $P_0, P_1, \ldots, P_m$, $m \geq 3$, with a curve consisting of m-2 cubic polynomial curve segments $Q_3, Q_4, \ldots, Q_m$



- The parameter range of Q_i is defined as $t_i \leq t \leq t_{i+1}$ for $3 \leq i \leq m$
- For each $i \geq 4$, there is a join point or knot between Q_{i-1} and Q_i at the parameter value t_i known as knot value.
- The initial and final points at t_3 and t_{m+1}
- So there are m+1 control points $(P_0, P_1, \ldots, P_m)$, m-2 segments $(Q_3, Q_4, \ldots, Q_m)$ and m-1 knots $(t_3, t_4, \ldots, t_m, t_{m+1})$

- The term uniform means the knots are spaced at equal intervals of the parameter t
  We can assume that $t_3 = 0$ and $t_{i+1} - t_i = 1$
- In a rational cubic polynomial curve, $x(t), y(t) \& z(t)$ are defined as ratio of two cubic polynomials
- The "B" stands for basis, since the spline can be represented as weighted sums of polynomial basis function
- Each of the $m-2$ curve segments of a B-spline curve is defined by four control points out of the $m+1$ control points
- The curve segment Q_i is defined by points $P_{i-3}, P_{i-2}, P_{i-1}$ & P_i. Thus the geometry vector for segment Q_i is

$$G_{BS_i} = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}, \quad 3 \leq i \leq m$$

Segment	Control Points	Parameter
Q_3	P_0, P_1, P_2, P_3	$t_3 = 0,\ t_4 = 1$
Q_4	P_1, P_2, P_3, P_4	$t_4 = 1,\ t_5 = 2$
Q_5	P_2, P_3, P_4, P_5	$t_5 = 2,\ t_6 = 3$
⋮		
Q_m	$P_{m-3}, P_{m-2}, P_{m-1}, P_m$	$t_m = m-3,\ t_{m+1} = m-2$

NOTE: To satisfy the convex hull property, the blending function should be non-negative and sum = 1.

The parametric equation is

$$Q(t) = T.M.G$$

The B-spline formulation for curve segment i is

$$Q_i(t) = T_i\, M_{BS}\, G_{BS_i}, \quad t_i \leq t \leq t_{i+1}$$

where $T_i = \left[(t-t_i)^3 \ (t-t_i)^2 \ (t-t_i) \ 1 \right]$

109

The B-Spline basis matrix, $M_{BS} = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$

The B-Spline blending function

$B_{BS} = T_i \cdot M_{BS}$

The blending function for each curve segment are exactly same, because for each segment i the values of $t - t_i$ range from 0 at $t = t_i$ to 1 at $t = t_{i+1}$

If we replace $t - t_i$ by t and the interval $[t_i, t_{i+1}]$ by $[0, 1]$

$$B_{BS} = T \cdot M_{BS} = [t^3 \; t^2 \; t \; 1] \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

$$= \left[\frac{-t^3+3t^2-3t+1}{6}, \; \frac{3t^3-6t^2+4}{6} \; \frac{-3t^3+3t^2+3t+1}{6} \; \frac{t^3}{6}\right]$$

$$Q(t) = B_{BS} \cdot G_{BSi} =$$

$$\Rightarrow Q(t-t_i) = \left[+\frac{(1-t)^3}{6} \; \frac{3t^3-6t^2+4}{6} \; \frac{-3t^3+3t^2+3t+1}{6} \; \frac{t^3}{6}\right]\begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}$$

$$= \frac{(1-t)^3}{6} P_{i-3} + \frac{3t^3-6t^2+4}{6} P_{i-2} + \frac{-3t^3+3t^2+3t+1}{6} P_{i-1} + \frac{t^3}{6} P_i$$

$0 \leq t \leq 1.$

Properties of B-Spline

1) Q_i and Q_{i+1} are C^0, C^1 & C^2 continuous where they join.

2) The sum of the B-Spline basis functions for any parameter value is 1.

3) The maximum order of the curve is equal to the number of vertices of defining polygon

4) The degree of B-Spline polynomial is independent on the number of vertices of defining polygon.

5) B-spline allows the local control over the curve surface because each vertex affects the shape of a curve only over a range of parameter values.

Computer Graphics

Topic:

Fractal Geometry

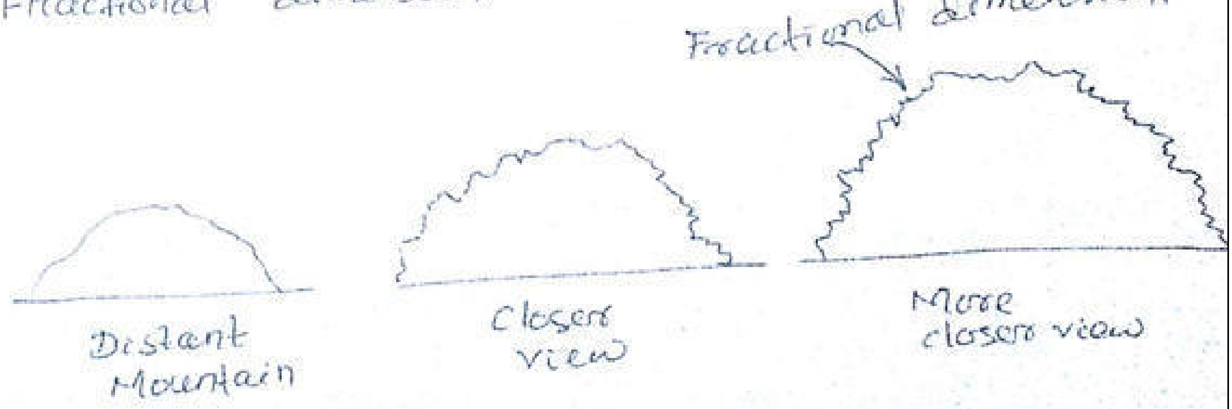
Contributed By:

Jasaswi Prasad Mohanty

FRACTAL GEOMETRY

Fractal: A fractal is defined as a rough or fragmented geometric shape that can be split into parts, each of which is approximately a reduced-size reproduction of the complete shape based on the property known as self-similarity.

- It was derived from the Latin word fractus which means "broken" or "fractured"
- Natural objects can be realistically described using fractal geometry methods. Eg: cloud, mountain, tree, stone etc
- Fractal methods use procedures rather than equations to model objects. So it uses procedural modelling.
- The major characteristic of any procedural model is that the model is not based on data, but rather on the implementation of a procedure following a particular set of rules.
- A fractal combines the following characteristics:
  a) Its parts have the same form or structure as a whole, except that they are at a different scale & may be slightly deformed.
  b) Its form is extremely irregular or fragmented, & remains so, whatever the scale of examination.
  c) It is formed by iteration i.e. the procedure is used repeatedly (recursively)
     Eg: if $P_0 = (x_0, y_0, z_0)$ is a selected initial position the successive levels $P_1 = F(P_0)$, $P_2 = F(P_1)$, ..., $P_n = F(P_{n-1})$ are generated by a transformation function F.
  d) Fractional dimension



111

Generating Fractals

The four most common techniques used for generating fractals are:

i) Escape-time fractals / Orbit fractals:
- It is defined by a formula or recurrence relation at each point in a space
- Eg: Mandelbort set, Julia set, Burning Ship fractal, Nova fractal, Lyapunov fractal

ii) Iterated function systems:
- These systems have a fixed geometric replacement rule
- Eg: Cantor set, Sierpinski carpet, Sierpinski gasket, Peano curve, Koch snowflake, Harter-Highway dragon curve, T-square, Menger sponge

iii) Random Fractals:
- The fractals are generated by stochastic rather than deterministic processes.
- Eg: Trajectories of the Brownian motion, Fractal land shapes, Brownian Tree

iv) Strange attractors:
- These are generated by iteration of a map or the solution of a system of initial-value differential equations that exhibit chaotic properties.

Fractal Classification

Fractals can be classified as:

1) Self-similar fractals:
- These fractals have parts that are scaled-down versions of the entire object.
- Starting with an initial shape, we construct the object subparts by applying a scaling parameter s

to the overall shape.

- If we apply random variations to the scaled-down subparts the fractal is said to be statistically self-similar.
- Eg: Tree, Shrub, Plants.

2) Self-affine fractals:
   - These fractals have parts that are formed with different scaling parameters sx, sy, sz in different coordinate directions.
   - Eg: Terrain, Waters, Clouds
   - If we apply random variations then that is known as statistically self-affine fractals.

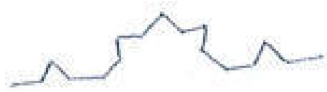
3) Invariant fractal sets:
   - These are formed with nonlinear transformations
   - This class of fractals includes:
     - Self-squaring fractals such as Mandelbrot set formed with squaring functions in complex spa[ce]
     - self-inverse fractals formed with inversion procedures.

Koch Curve / Von Koch Snow Flake

1) Start with a straight line of length 1.

2) The straight line is divided into 3-equal parts and middle part is replaced by two linear segments at angles 60° and 120°

3) Repeat step-1 & 2 to the 4 line segments generated in step-2.



4) Further iteration will generate the following curves:

It can be iterated an infinite number of times by dividing a straight line segment into three equal parts and substituting the intermediate part with two segments of the same length.

Fractal Dimension

- The amount of variation in the structure of a fractal object is described as the fractal dimension, D.
- The fractal dimension D for self-similar objects can be obtained from

$$ns^D = 1$$ where D: fractal dimension, n: no. of sub-parts, s: scaling factor

$$\Rightarrow n = \frac{1}{s^D} = \left(\frac{1}{s}\right)^D$$

$$\Rightarrow \ln n = D \cdot \ln\left(\frac{1}{s}\right)$$

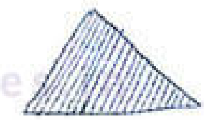
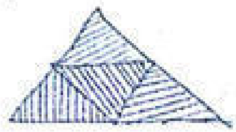
$$\Rightarrow D = \frac{\ln n}{\ln(1/s)}$$

NOTE: Fractal dimension is a measure of the roughness or fragmentation of the object. More jagged-looking objects have larger fractal dimensions.

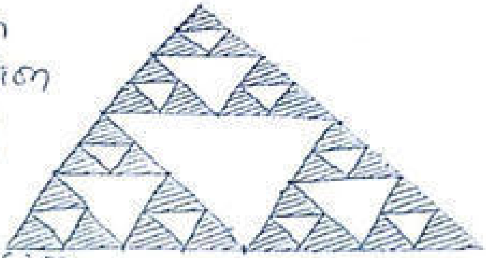
Eg: For koch curve
$n = 4$, $s = 1/3$

$$D = \frac{\ln 4}{\ln(1/1/3)} = \frac{\ln 4}{\ln 3} = 1.262$$

Sierpinski Triangle

1. Start with an equilateral triangle 
2. Connect the midpoints of each side of the triangle to form four separate triangles.
3. Cut out the triangle in the centre. 
4. Repeat the steps 1-3 on the  three ~~blanks~~ black triangles left behind.
The centre triangle of each black triangle at the corners were cut out as well.

5. Further repetition with adequate screen resolution will give the following pattern.



Calculation of fractal dimension

Divide each side by 2
- Makes 4 triangles
- We keep 3

So $n = 3$

$s = 1/2$

Fractal dimension; $D = \frac{\ln 3}{\ln\left(\frac{1}{1/2}\right)} = \frac{\ln 3}{\ln 2} = 1.585$

Computer Graphics

Topic:

Three Dimensional Geometric And Modeling Transformations

Contributed By:
Jasaswi Prasad Mohanty

Three-Dimensional Geometric Transformations

Translation

A point is translated from position $P(x,y,z)$ to $P'(x',y',z')$ with the following matrix operation

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$ (In cartesian co-ordinate system)

or

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ (In Homogeneous form)

Scaling

A point is scaled from position $P(x,y,z)$ to $P'(x',y',z')$ with the following matrix operation

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$ (In cartesian co-ordinate system)

or

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ (In Homogeneous form)

Scaling w.r.t. fixed point (x_f, y_f, z_f)

$$x' = x_f + (x - x_f) S_x$$
$$y' = y_f + (y - y_f) S_y$$
$$z' = z_f + (z - z_f) S_z$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} 1-S_x & 0 & 0 \\ 0 & 1-S_y & 0 \\ 0 & 0 & 1-S_z \end{bmatrix} \begin{bmatrix} x_f \\ y_f \\ z_f \end{bmatrix}$$

[In Cartesian co-ordinate system]

In homogeneous form

(Translate back to (x_f, y_f, z_f)) · (scale) · (Translate (x_f, y_f, z_f) to origin)

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

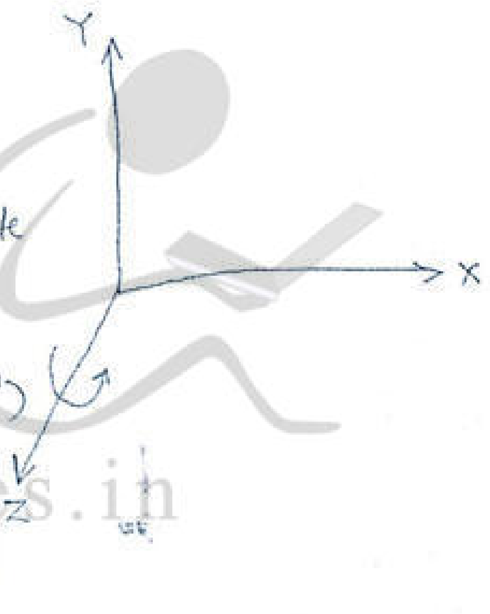
$$= \begin{bmatrix} 1 & 0 & 0 & x_f \\ 0 & 1 & 0 & y_f \\ 0 & 0 & 1 & z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_f \\ 0 & 1 & 0 & -y_f \\ 0 & 0 & 1 & -z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

Rotation

Rotation about z-axis:

- If the rotation is carried out about the z-axis, the z-coordinate remain unchanged because rotation occurs in the XY plane



If $P(x,y,z)$ is rotated to $P'(x',y',z')$ with an angle θ then

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

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

$$P' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R_z(\theta) \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = R_z(\theta) \cdot P$$

In cartesian co-ordinate system

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

$$P' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = R_z(\theta) \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = R_z(\theta) \cdot P$$

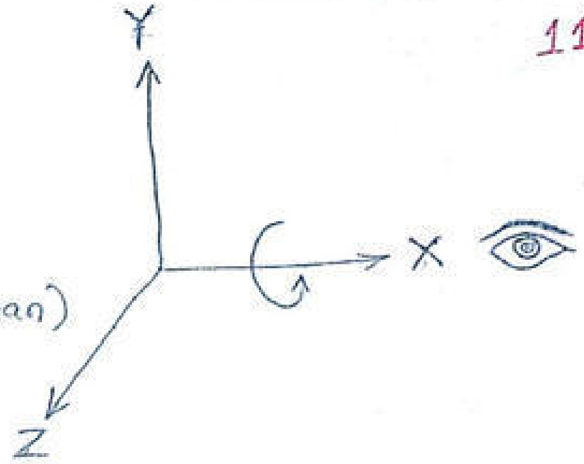
In homogeneous co-ordinate system.

Rotation about X-axis

$x' = x$

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

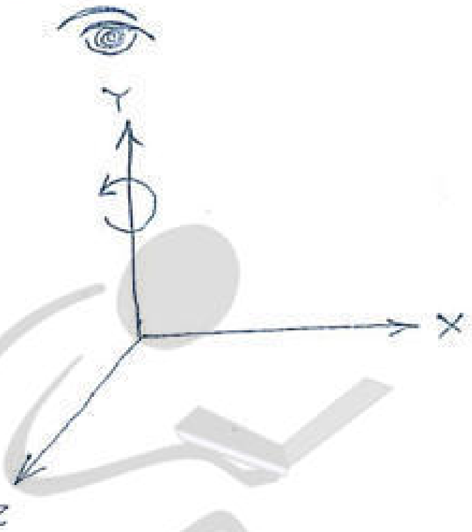
$z' = y\sin\theta + z\cos\theta$



$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \text{(Cartesian)}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{(Homogeneous)}$$

Rotation about Y-axis



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

$y' = y$

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

$$R_Y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \text{(Cartesian)}$$

$$R_Y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{(Homogeneous)}$$

Rotation about an Arbitrary Axis

Let us assume that we want to perform a rotation by θ degree about an axis in space passing through the point (x_0, y_0, z_0) with direction cosines (c_x, c_y, c_z).

To do this follow the following steps:

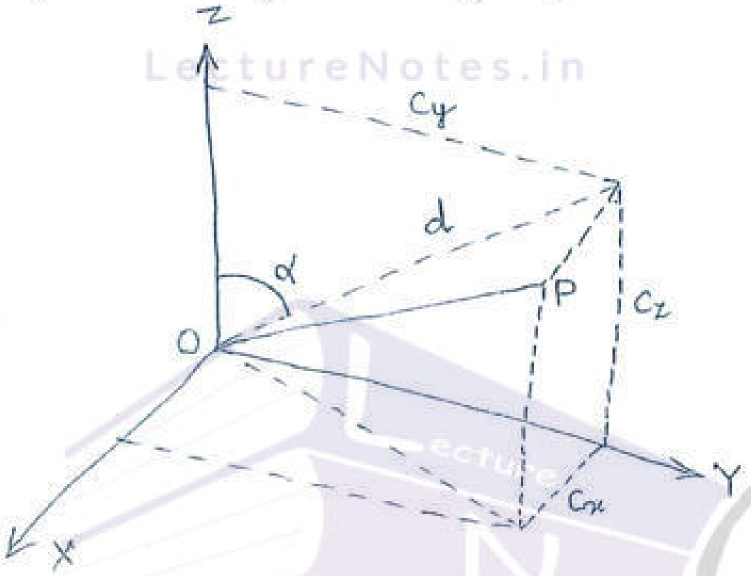
1. Translate by $|T| = -\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$

2. Rotate the axis into one of the principle axes.
   Let us pick, $Z(|R_x|, |R_Y|)$

3. Rotate by θ degrees in z ($|R_z(\theta)|$)
4. Undo the rotations to align the axis
5. Undo the translation: Translate by $\begin{bmatrix} +x_0 \\ +y_0 \\ +z_0 \end{bmatrix}$

Step 2 is going to take 2 rotations:

i) About x-axis (to place the axis in the xz plane)
ii) About y-axis (to place the result coincident with the z-axis)



Rotation about x-axis by α: How to find α?

- We have projected the axis OP onto ZY plane. Here c_x, c_y, c_z are the direction cosines and OP is the unit vector in length.
- d is the diagonal i.e. projection of OP on ZY plane
- So if we rotate OP by an angle α to place it on XZ plane is equivalent to rotating d such that it coincides with the z-axis

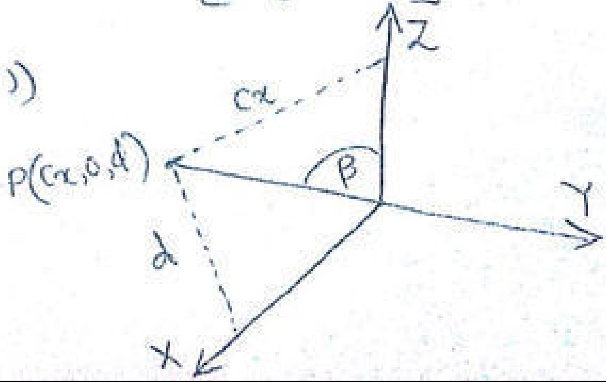
$d = \sqrt{c_y^2 + c_z^2}$

$\cos\alpha = c_z/d$

$\sin\alpha = c_y/d$

So $\alpha = \sin^{-1}\left[\frac{c_y}{\sqrt{c_y^2 + c_z^2}}\right]$

So after first rotation (Step 2(i)) we have the following diagram



Now we have to rotate OP by β about Y-axis

How to find β?

- Determine the angle β to rotate the result into z-axis:

  The x component is C_x and z component is d.

  $\cos\beta = d / (\text{length of the unit vector}) = d$

  $\sin\beta = C_x / (\text{length of the unit vector}) = C_x$

So final transformation matrix for 3D rotation about an arbitrary axis:

$$M = |T| . |R_X| \, |R_Y| . |R_Z| \, |R_Y|^{-1} \, |R_X|^{-1} \, |T|^{-1}$$

where:

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C_z/d & -C_y/d & 0 \\ 0 & C_y/d & C_z/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_Y = \begin{bmatrix} d & 0 & -C_x & 0 \\ 0 & 1 & 0 & 0 \\ C_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_Z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = [T R_X R_Y] [R_Z] [T R_X R_Y]^{-1}$$
$$= C . [R_Z] C^{-1}$$

Any coordinate position P can be transformed with the following sequence:

$$P' = P . |T| . |R_X| \, |R_Y| \, |R_Z| \, |R_Y|^{-1} |R_X|^{-1} |T|^{-1}$$

Rotation of an object rotated about an axis parallel to one of the coordinate axes:

We can attain the desired rotation with the following transformation sequences:

1. Translate the object so that the rotation axis coincides with the parallel coordinate 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.

Any coordinate position P is transformed as

$$P' = T^{-1} R_X(\theta) . T . P$$

[Assuming axis parallel to x-axis]

NOTE: If you are given 2 points on the axis of rotation you can calculate the direction cosines of the axis as follows:

$$V = [(x_1 - x_0) \;\; (y_1 - y_0) \;\; (z_1 - z_0)]$$

$$C_x = \frac{x_1 - x_0}{|V|}$$

$$C_y = \frac{y_1 - y_0}{|V|}$$

$$C_z = \frac{z_1 - z_0}{|V|}$$

where $|V|$ is the length of the vector V

$$|V| = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$$

Reflection

The transformation matrices

$$T_{XY} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad T_{YZ} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad T_{ZX} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

produce reflection about XY, YZ and ZX planes respectively

Reflection through an arbitrary plane

Method is similar to that of rotation about an arbitrary axis

$$M = |T| \cdot |R_x||R_y||R_{fl}||R_y|^{-1}|R_x|^{-1}|T|^{-1}$$

T does the job of translating the origin to the plane
R_x and R_y will rotate the vector normal to the reflection plane (at the origin), until it is coincident with the Z-axis.
R_{fl} is the reflection matrix about XY plane or Z=0 plane

Scaling

Scaling transformation of a position $P = (x, y, z)$ relative to the origin can be written as

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

or $P' = S \cdot P$

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

1. Translate the fixed point to the origin
2. Scale the object relative to the coordinate origin
3. Translate the fixed point to its original position

So $S = T(x_f, y_f, z_f) \cdot S(s_x, s_y, s_z) \cdot T(-x_f, -y_f, -z_f)$

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

NOTE:

1) We can preserve the original shape of an object with a uniform scaling $(s_x = s_y = s_z)$.
2) We can form the inverse scaling matrix by replacing the scaling parameters s_x, s_y, s_z by their reciprocals.

Shear

- In 3D we can generate shears relative to x, y or z axis
- Here we can push in two coordinate axis direction by keeping the third one fixed.
- The transformation matrix in both x, y directions keeping the z-coordinate same is:

$$SH_{xy} = \begin{bmatrix} 1 & 0 & g & 0 \\ 0 & 1 & h & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Here a point $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ is transformed to $\begin{bmatrix} x+gz \\ y+hz \\ z \\ 1 \end{bmatrix}$

- By replacing g by −g and h by −h the same point will be transformed to $\begin{bmatrix} x-gz \\ y-hz \\ z \\ 1 \end{bmatrix}$
- The shear transformation in xz direction is

$$SH_{xz} = \begin{bmatrix} 1 & d & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- The shear transformation in yz direction is

$$SH_{yz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ b & 1 & 0 & 0 \\ c & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation Matrix in 3D : Summary

$$A = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & i & j & r \\ l & m & n & s \end{bmatrix} = \begin{bmatrix} T & K \\ \Gamma & \theta \end{bmatrix}$$ where

$T = \begin{bmatrix} a & b & c \\ d & e & f \\ g & i & j \end{bmatrix}$ produces linear transformations: scaling, shearing, reflection and rotation.

$K = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$ produces translation

$\Gamma = [\, l \;\; m \;\; n \,]$ yields perspective transformation

$\theta = s$ is responsible for uniform scaling.

Computer Graphics

Topic:

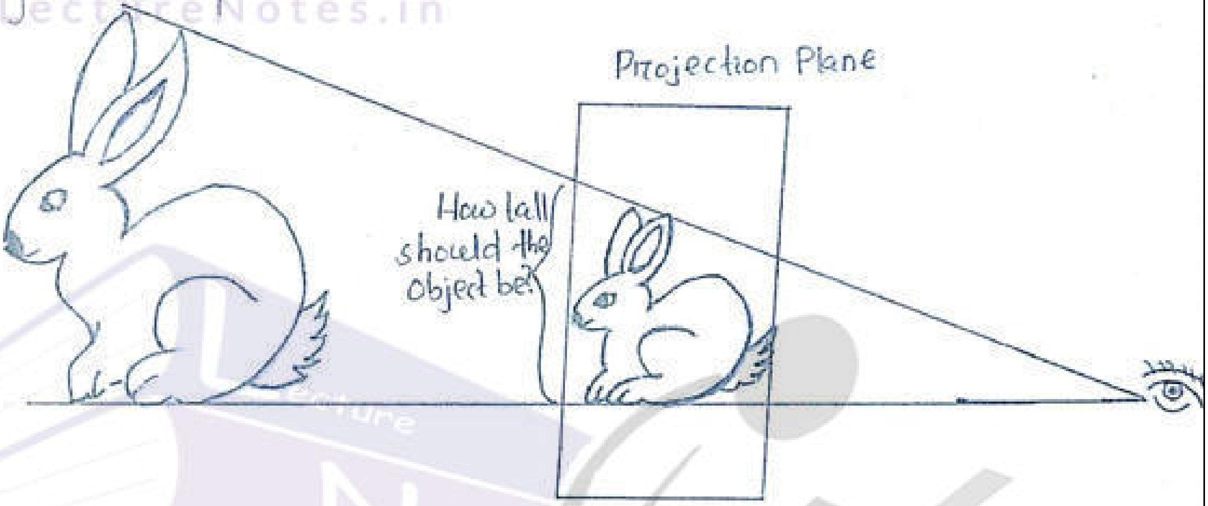
Projections

Contributed By:

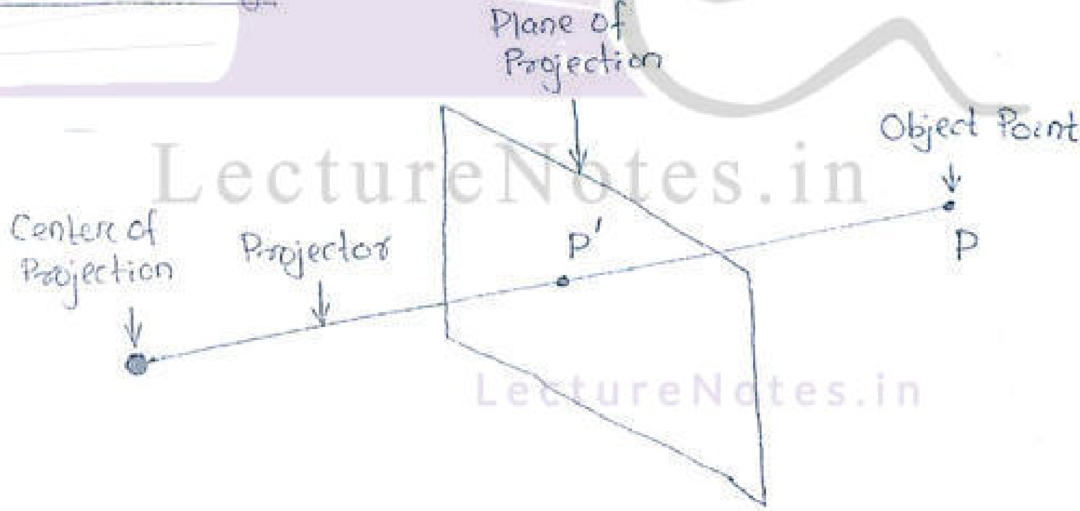
Jasaswi Prasad Mohanty

PROJECTION

- 3D projection is any method of mapping 3D points to a 2D plane.
- Projection of a 3D object is defined by straight projection rays (projectors) emanating from the center of projection (COP) that represents a camera or viewing position, passing through each point of the object and intersecting the projection plane

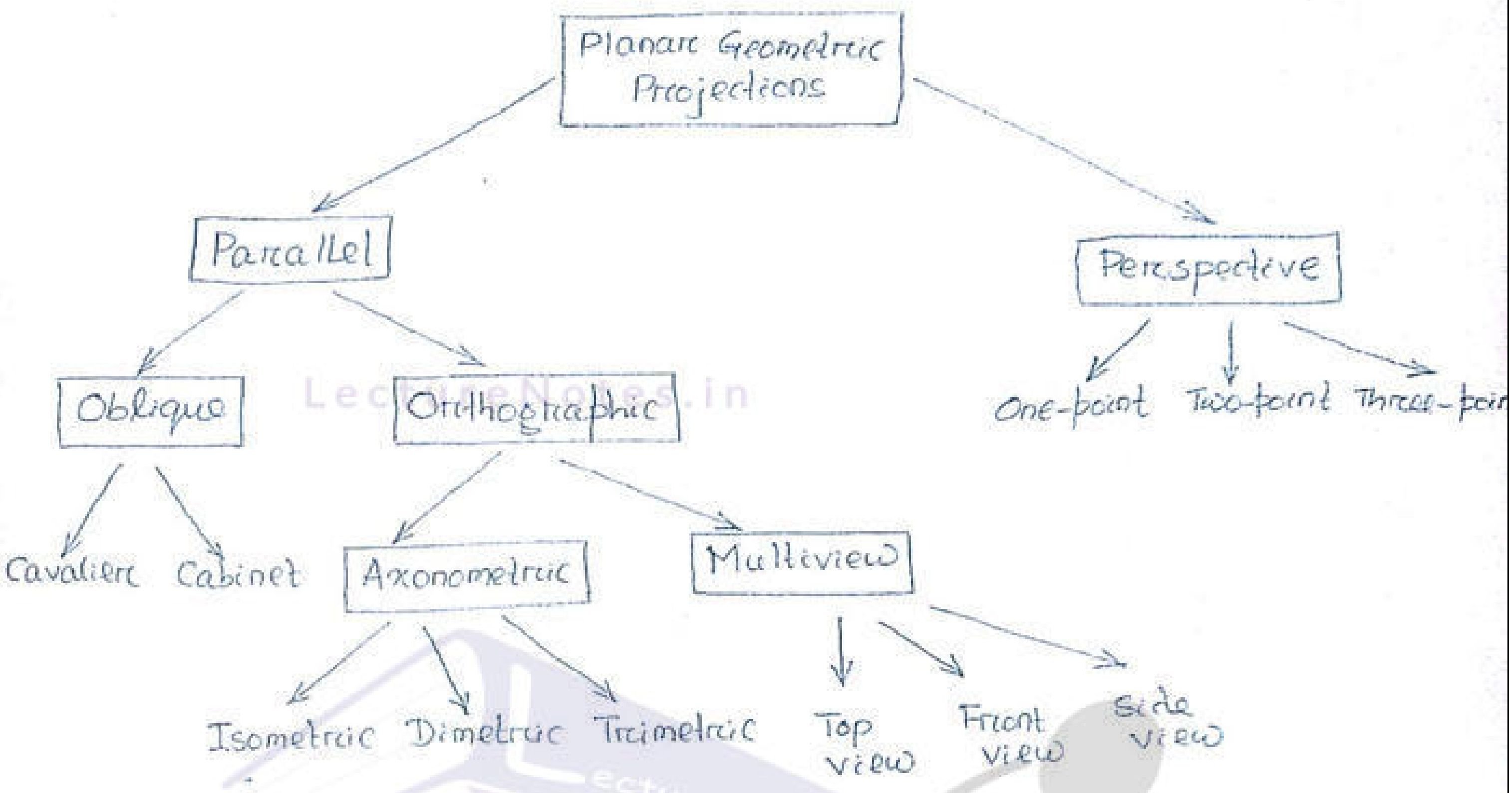


Projection Geometry



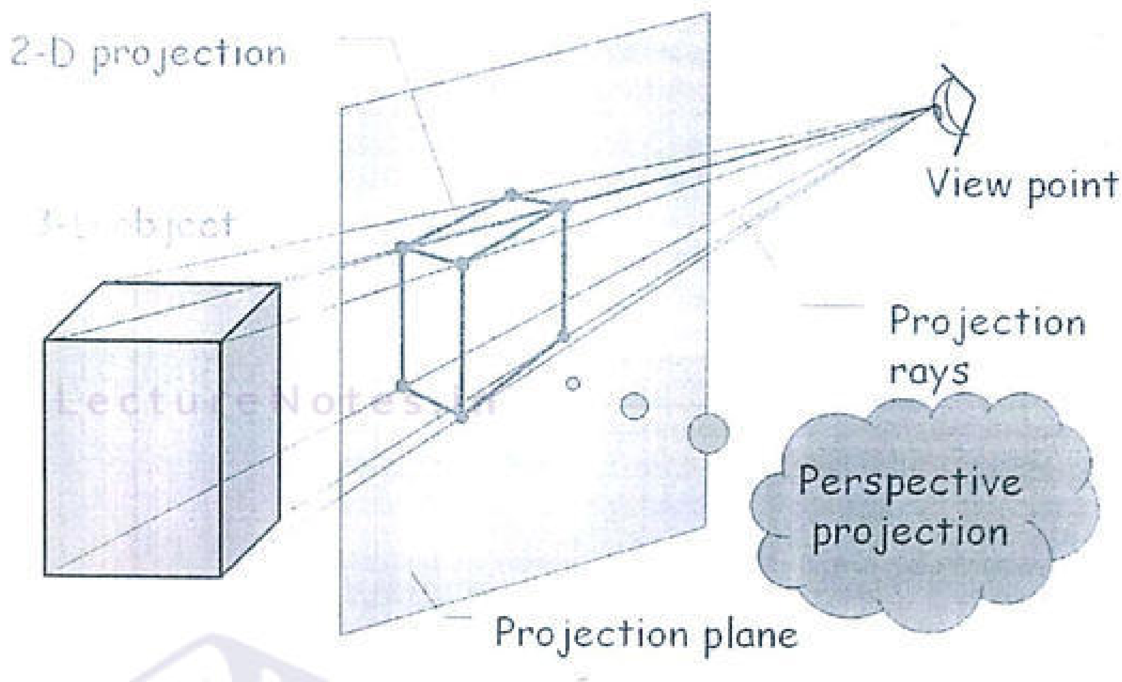
P(3D) ⇒ P'(2D)

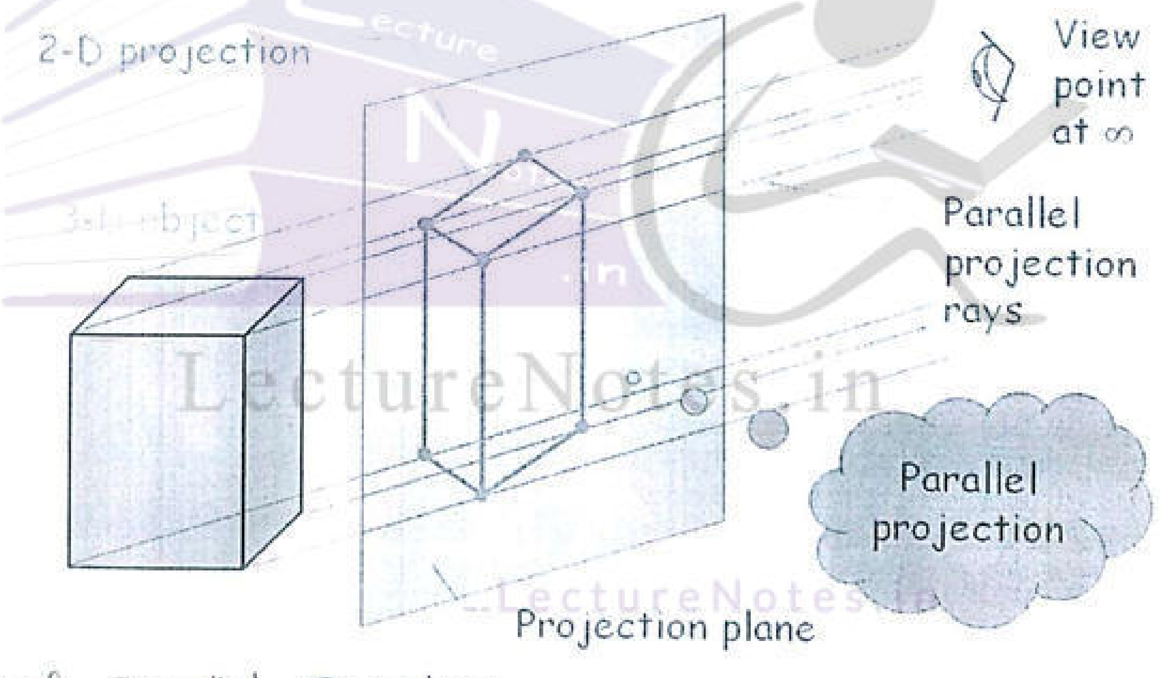
Classification of Geometric Projections



Parallel vs Perspective Projection

Parallel Projection	Perspective Projection
1) Less realistic looking	1) Look realistic
2) Centre of projection is at infinity.	2) Centre of projection is at a finite point
3) Projectors are parallel to each others	3) Projectors intersect at COP
4) Object size does not change	4) Size varies inversely with distance
5) Useful in applications requiring the relative proportions of an object to be maintained	5) Distance and angles are not preserved.





Types of Parallel Projection

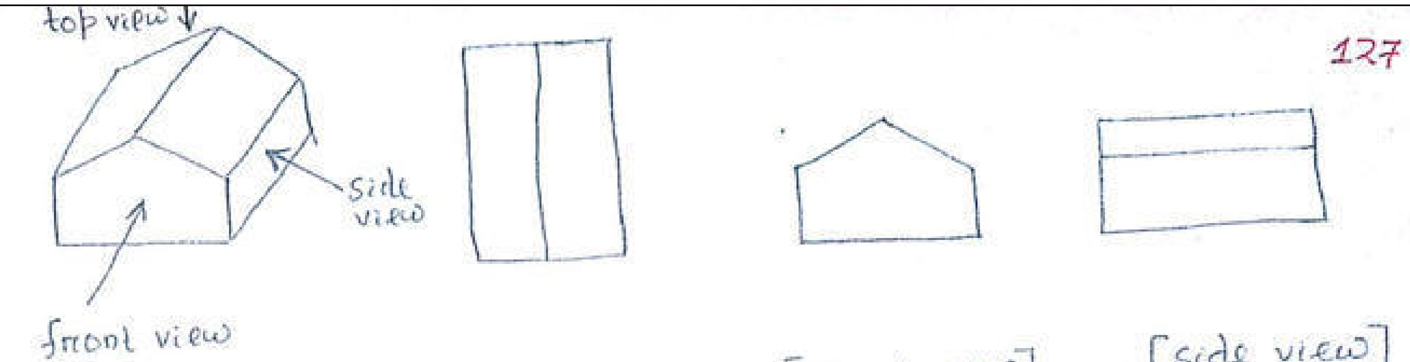
1. Orthographic Projection: If the direction of projection is perpendicular to the projection plane, it is an orthographic projection
2. Oblique Projection: If the direction of projection is not perpendicular to the projection plane, it is an oblique projection.

Orthographic Projection

- It is the simplest parallel projection technique used in engineering drawing.
- It is frequently used in architectural design and computer aided design (CAD).
- All objects that have the same dimensions appear the same size, regardless of whether they are far away or nearby.
- It is a projection on any one of the coordinate planes $x=0$, $y=0$ or $z=0$.
- The transformation matrix for orthographic projection of object onto $x=0$, $y=0$, and $z=0$ planes from COP at infinity on respective axes are given by P_x, P_y, & P_z.

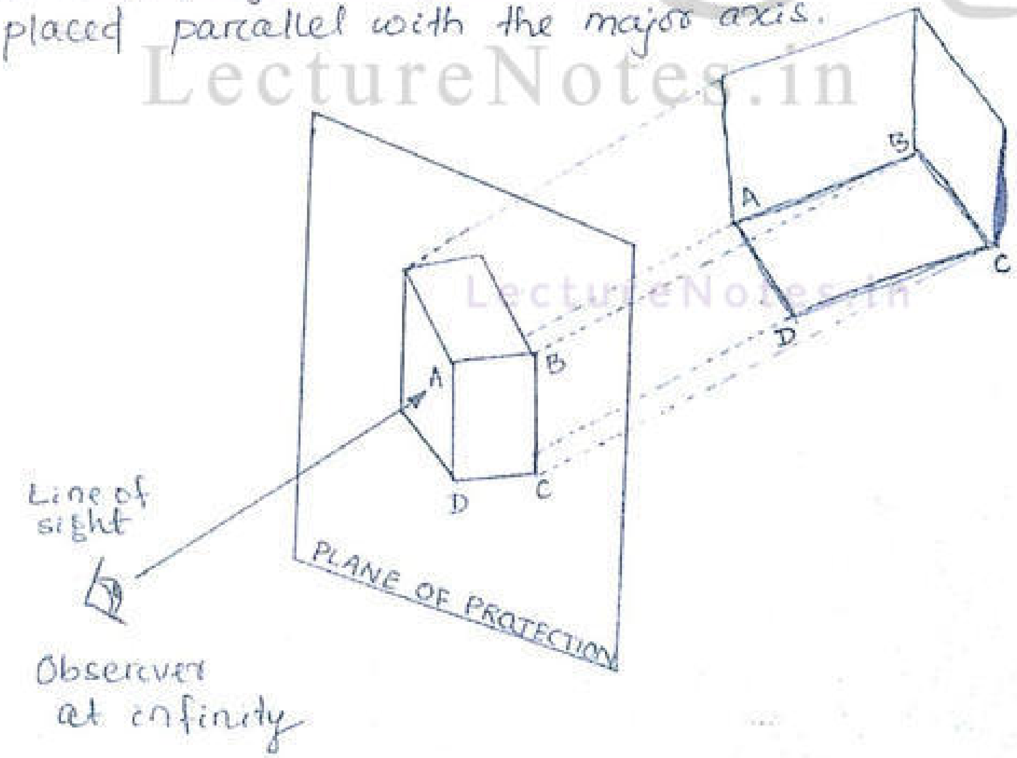
$$[P_x] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad [P_y] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad [P_z] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Orthographic projections are further categorized as
  - Multiview
  - Axonometric
- Multiview projection displays a single face of a 3D object.
- Front, side, and top or planar view are common type of multiview projections.
- In top view the view-plane normal is parallel ~~to the~~ with positive y-axis, and the x and z coordinates for each point are mapped to the view-plane.
- In front view the view-plane normal is parallel with positive z-axis, the x and y coordinates for each point are mapped to the view-plane.
- Similarly in side view the view-plane normal is parallel with positive x-axis, the y & z coordinates for each point are mapped to the view-plane.
- Multiview projections are often used in engineering and architectural drawing.



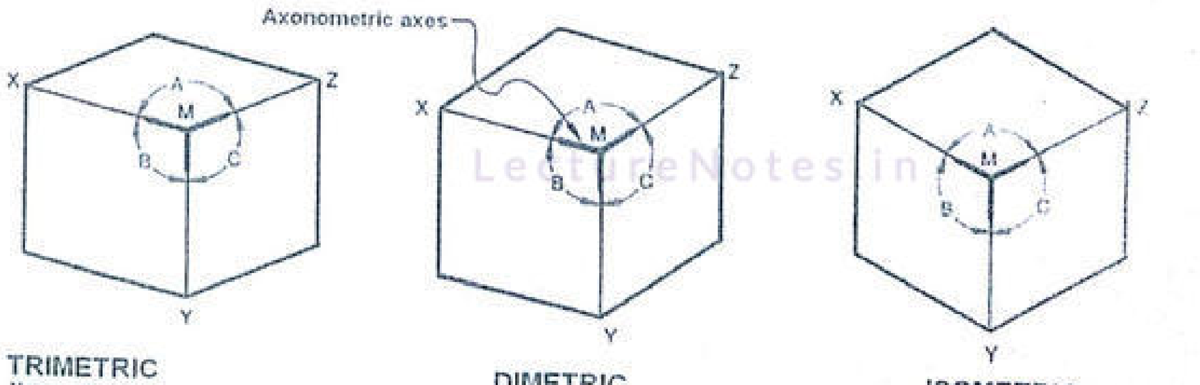
[3D view of an object] [Top view] [Front view] [side view]

- Axonometric projection is a parallel projection technique used to create pictorial drawing of an object by rotating the object along one or more of its axes relative to the plane of projection
- Difference between multiview and axonometric projection:
    - In multiview only two dimensions of an object are visible in each view and hence more than one view is required to define the object.
    - In axonometric projection, the object is rotated about an axis to show all three dimensions and only one view is required.
- The plane of projection for an axonometric view is usually not placed parallel with the major axis.



- The direction of the projection plane normal positions the projection plane such that it intersects at least two of the major axes.

- Axonometric projections are classified into three classes:
    - Isometric Projection
    - Dimetric Projection
    - Trimetric Projection
- In trimetric projection, the direction of viewing is such that all of the three axes of space appear unequally foreshortened. The scale along each of the three axes and the angles among them are dictated by the angle of viewing. Trimetric projection is rarely used.
- In dimetric projection, the direction of viewing is such that two of three axes of space appear equally shortened. Here two of the three angles among the axes are equal. Dimetric drawing are less pleasing to the eye, but are easier to produce than trimetric drawing.
- In isometric projection, the most commonly used form of axonometric projection in engineering drawing. Here all three angles are equal. The isometric is the least pleasing to the eye, but is the easiest to draw and dimension.



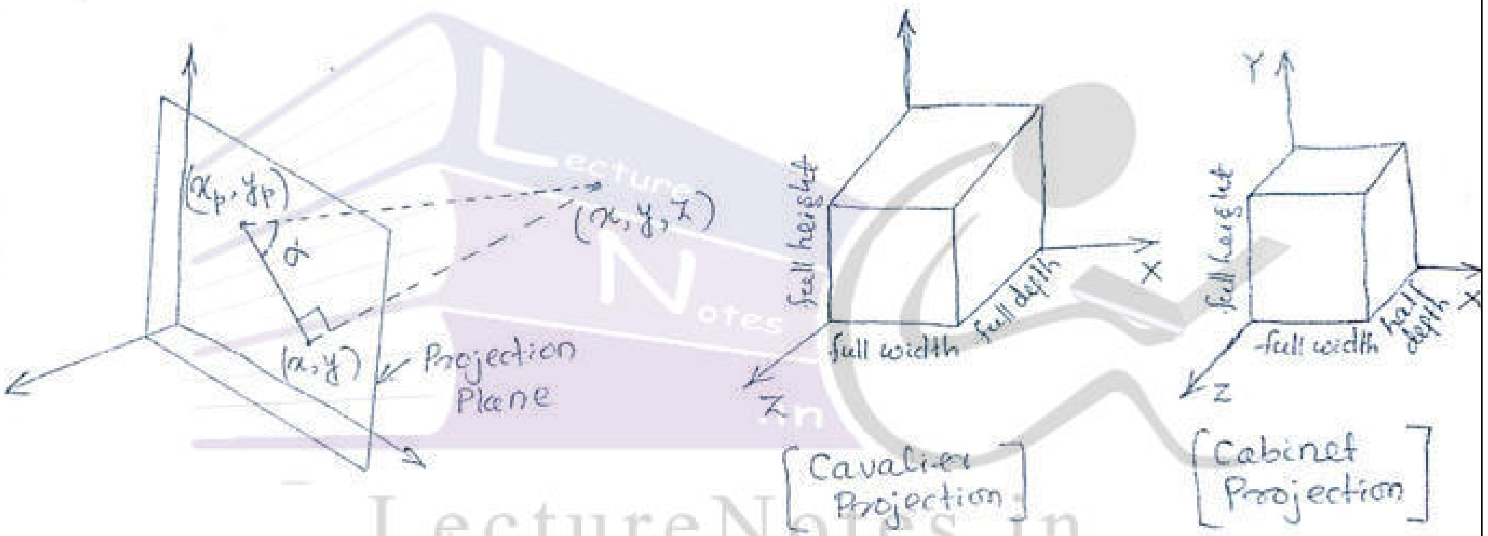
TRIMETRIC
No equal angles
No equal corners

DIMETRIC
Angles A & C are equal
Corners MY and MX are equal in length

ISOMETRIC
Angles A, B, & C are equal
Coreners MZ,MY, & MX are equal length

Oblique Projection

- In oblique projections, parallel projection rays are not perpendiculas to the viewing plane. They strike the projection plane at an angle other than 90°.
- Because of its simplicity, an oblique projection is used exclusively for pictorial purposes rather than for formal working drawing.
- In an oblique projection, the displayed angles among the axes, as well as foreshortening factors (scale) are arbitrary.
- Special types of oblique projections are:
  - ✓ cavalier projection
  - ✓ cabinet projection



- In the above figure

  (x_p, y_p) : Projection of point (x, y, z) on projection plane by oblique projection

  (x, y) : Projection of point (x, y, z) on projection plane by orthographic projection

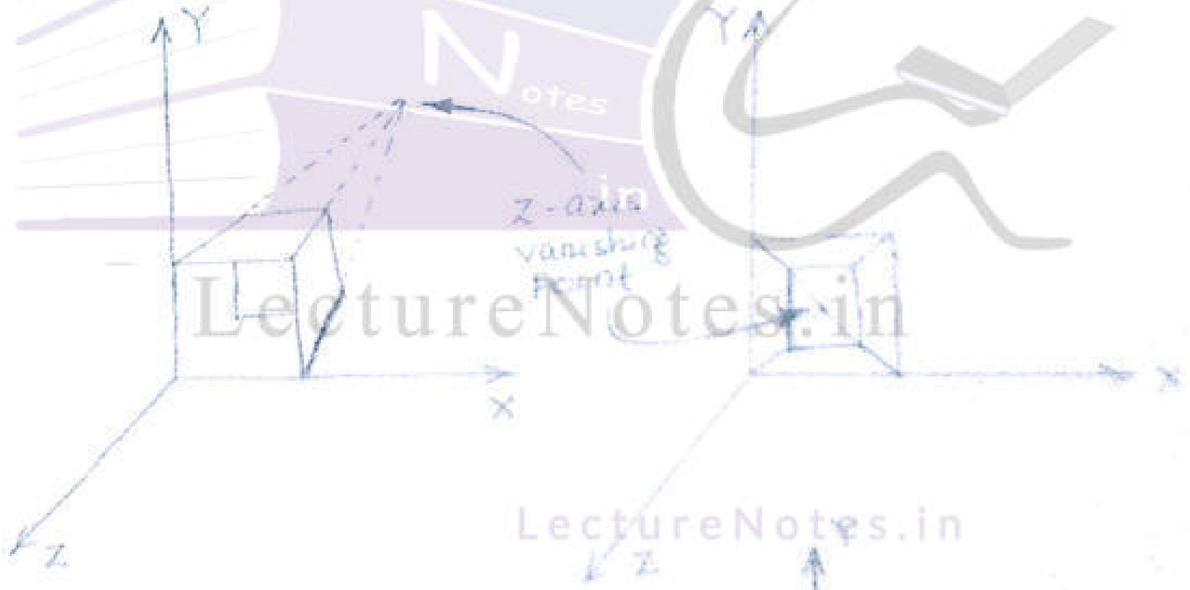
  α : Angle between the line from (x, y, z) to (x_p, y_p) and the line from (x_p, y_p) to (x, y)

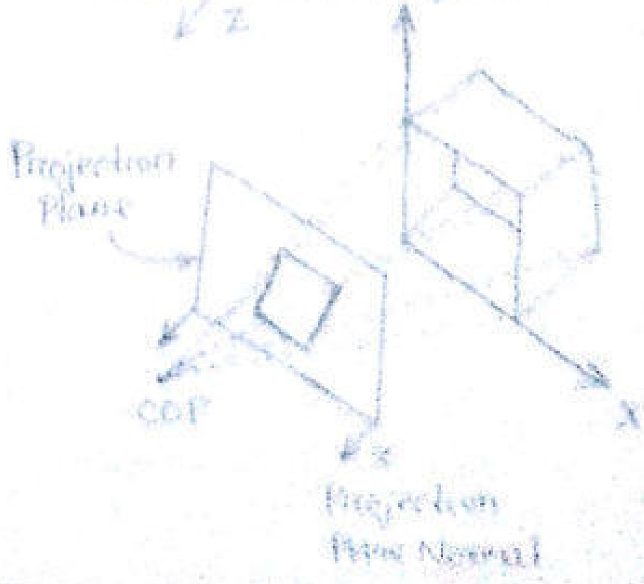
- When $\alpha = 45°$, the projection is cavalier projection. Here a cube will be displayed with all sides maintaining equal lengths. The advantage is edges can be measured directly. However, cavalier projection can make an object look too elongated.

- When $\alpha = 63.4°$, the projection is labeled as a cabinet projection. For this angle, lines perpendicular to the view-plane is displayed one-half the actual length. Cabinet projections appear more realistic than cavalier projections.

Perspective Projection

- Perspective projection is a type of projection where 3D objects are not projected along parallel lines, but along lines emerging from a single point.
- The lines parallel in nature appear to intersect on the projected image.
- The perspective projections of any set of parallel lines that are not parallel to the projection plane converge to a point known as vanishing point.





- The number and placement of the vanishing points determine which perspective technique is being used.
- In perspective transformation, object size is reduced with increasing distance from the COP and non-uniform foreshortening of lines in the object as a function of orientation and the distance of the object from COP occurs.
- Perspective foreshortening is the illusion that objects and lengths appear smaller as there distance from COP increases.
- A perspective projection is done in two steps : perspective transformation and parallel projection.
  - ✓ A perspective transformation converts a 3D object into a deformed 3D object.
  - ✓ A parallel projection project the object onto a 2D image plane.
- Of the many types of perspective drawing, the most common categorizations of artificial perspective are :
  - ✓ One-point perspective
  - ✓ Two-point perspective
  - ✓ Three-point perspective

  These categories refer to the number of vanishing points in the perspective drawing.
- In One-point perspective the projection plane is perpendicular to one of the principal axes. In two-point the projection plane intersect exactly two principal axes and in three-point the projection plane is not parallel to any principal axes.

One-point perspective

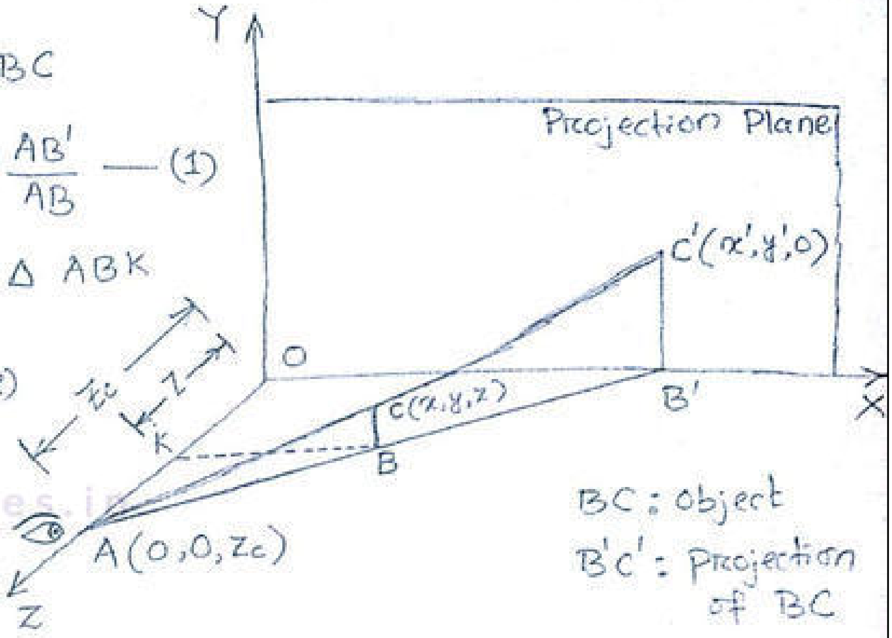
- Let us consider that we want to project an object BC onto a projection plane parallel to x & y axes and perpendicular to z-axis.
- Let the COP is on the z-axis at a distance of Z_c from the origin.
- The co-ordinate of point C is (x, y, z).

In $\Delta AB'C'$ and ΔABC

$$\frac{y'}{AB'} = \frac{y}{AB} \Rightarrow \frac{y'}{y} = \frac{AB'}{AB} \quad \text{--- (1)}$$

Similarly in $\Delta AB'O$ and ΔABK

$$\frac{Z_c}{Z_c - Z} = \frac{AB'}{AB} \quad \text{--- (2)}$$



From equ^n (1) & (2)

$$\frac{y'}{y} = \frac{Z_c}{Z_c - Z}$$

$$\Rightarrow y' = \frac{y}{1 - Z/Z_c}$$

In the similar way by comparing x values

$$\frac{x'}{x} = \frac{Z_c}{Z_c - Z} \Rightarrow x' = \frac{x}{1 - Z/Z_c}$$

So the co-ordinate of C' would be $\left(\frac{x}{1 - Z/Z_c}, \frac{y}{1 - Z/Z_c}, 0\right)$

The transformation matrix for perspective projection, T is obtained as the product of transformation matrix for orthographic projection (parallel projection) with transformation matrix for perspective transformation.

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{Z_c} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{Z_c} & 1 \end{bmatrix}$$

If the co-ordinate of a point is (x, y, z) then its coordinate after perspective projection would be

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{Z_c} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 1 - \frac{Z}{Z_c} \end{bmatrix} = \begin{bmatrix} x / 1 - Z/Z_c \\ y / 1 - Z/Z_c \\ 0 \\ 1 \end{bmatrix}$$

Let $\pi = -1/z_c$

So now the transformation matrix for perspective projection along z-axis is:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \pi & 1 \end{bmatrix}$$

Here projection plane is perpendicular to z-axis, COP is at $(0,0,z_c)$ and vanishing point (opposite to COP in the same distance from COP to origin) is at $(0,0,-z_c)$

Similarly the perspective projection along x-axis, where the projection plane is $x=0$, COP is at $(x_c,0,0)$ and vanishing point is at $(-x_c,0,0)$ is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ p & 0 & 0 & 1 \end{bmatrix} \quad \text{where } p = -\frac{1}{x_c}$$

The perspective projection along y-axis where the projection plane is $y=0$, COP is at $(0,y_c,0)$ and vanishing point is at $(0,-y_c,0)$ is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & q & 0 & 1 \end{bmatrix} \quad \text{where } q = -\frac{1}{y_c}$$

Two-point perspective transformation

- Two-point perspective can be used to draw an object as a one-point perspective, rotated for example, looking at the corner of a house, or looking at two forked roads shrink into the distance.
- Objects drawn in a two-point perspective have a more natural look.

- Two-point perspective transformation has two centre of projections and vanishing points.
- If the COP are at $(-1/p, 0, 0)$ and $(0, -1/q, 0)$ on x-axis and y-axis and the projection is on $z=0$ plane, the transformation matrix is $= \begin{bmatrix} 1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&1 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&0\\0&1&0&0\\0&0&1&0\\-\frac{1}{x_c}&-\frac{1}{y_c}&0&1 \end{bmatrix}$

$$= \begin{bmatrix} 1&0&0&0\\0&1&0&0\\0&0&0&0\\p&q&0&1 \end{bmatrix}$$

The vanishing points are $(1/p\ 0\ 0)$ and $(0, 1/q, 0)$ where $p = -1/x_c$, $q = -1/y_c$
~~where $p = -1/z_c$, $q =$~~

- Similarly if the COP are at $(-1/p, 0, 0)$ and $(0, 0, -1/r)$ on x and z-axis and the projection is on $y=0$ plane, the transformation matrix is

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

The vanishing points are $(1/p, 0, 0)$ and $(0, 0, 1/r)$ where $p = -1/x_c$, $r = -1/z_c$

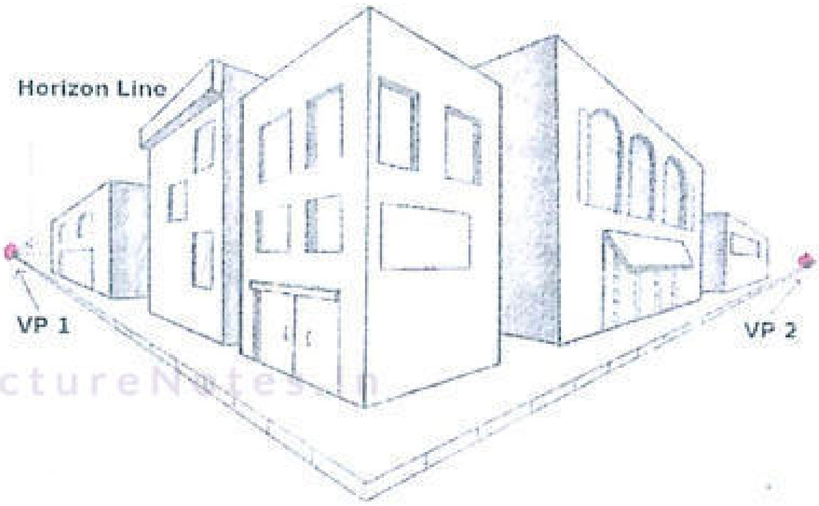
- If the COP are at $(0, -1/q, 0)$ and $(0, 0, -1/r)$ on y and z-axis and the projection is on $x=0$ plane, the transformation matrix is

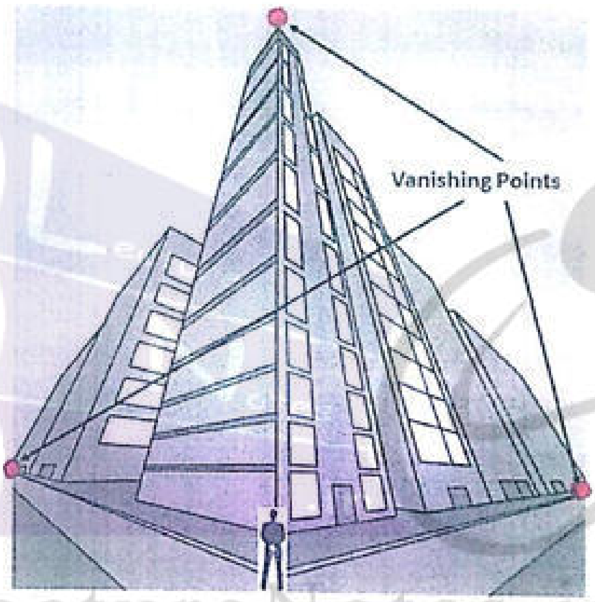
$$\begin{bmatrix} 1&0&0&0\\0&1&0&0\\0&0&0&0\\0&q&r&1 \end{bmatrix}$$

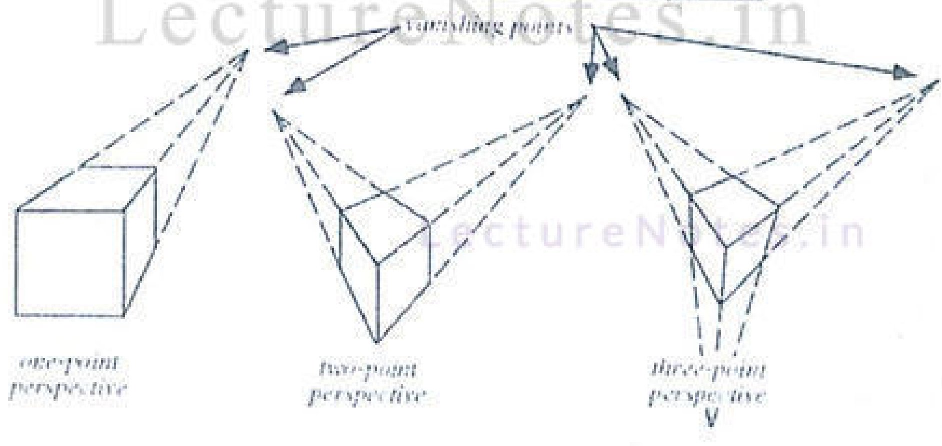
The vanishing points are $(0, 1/q, 0)$ and $(0, 0, 1/r)$ where $q = -1/y_c$, $r = -1/z_c$

Three-point perspective

- A three-point perspective is usually used for building seen from above (or below).
- The third vanishing point is either below the ground or in the sky (high in space)


Horizon Line
VP 1
VP 2


Vanishing Points


vanishing points
one-point perspective
two-point perspective
three-point perspective

136

- Here the three COPs are at $(-1/p, 0, 0)$ on x-axis, $(0, -1/q, 0)$ on y-axis and $(0, 0, -1/r)$ on z-axis. The three vanishing points are on x-axis at $(1/p, 0, 0)$, on y-axis at $(0, 1/q, 0)$ and on z-axis at $(0, 0, 1/r)$

- The transformation matrix for the projection plane z=0

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & q & r & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ p & q & r & 1 \end{bmatrix}$$

↗ transformation matrix for orthographic projection onto z=0 plane

↑ transformation matrix for three-point perspective transformation

↑ Final transformation matrix for three-point perspective projection

Computer Graphics

Topic:

Visible Surface Detection Methods

Contributed By:

Jasaswi Prasad Mohanty

Visible - Surface Detection Methods

- Visible-surface detection or hidden-surface elimination is the process of identifying which parts of a scene are visible from a chosen viewing position.
- There are many methods for visible-surface detection. Some methods require more memory, some involve more processing time, and some apply only to special type of objects.
- Deciding a method for a particular application depends on factors such as the complexity of the scene, type of objects to be displayed, available equipment, and whether static or animated displays are to be generated
- Broadly visible-surface detection algorithms are classified as:
  - ✓ Object-space methods: Compares objects and parts of objects to each other within the scene definition to determine which surfaces as a whole we should label as visible.
  - ✓ Image-space method: Visibility is decided point by point at each pixel position on the projection plane.

Back-Face Detection

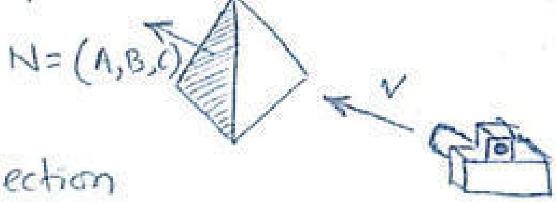
- It is a fast and simple object-space method for identifying the back faces of a polyhedron which is based on the inside-outside tests.
- A point (x,y,z) is inside a polygon surface with parameter A, B, C, D if

$$Ax + By + Cz + D < 0$$

- When an inside point is along the line of sight to the surface, the polygon must be a back face (we are inside that face and can't see the front of it from our viewing position)

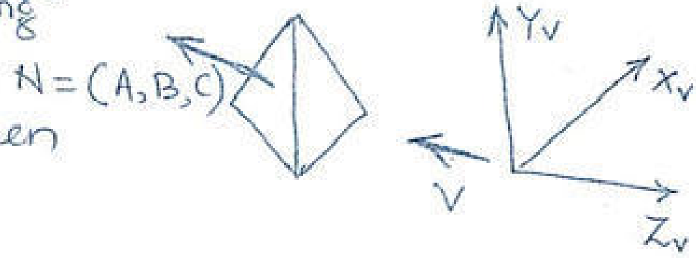
- Testing: Consider the normal vector N to a polygon surface, which has cartesian components (A, B, C). V is a vector in the viewing direction from the camera. The polygon is backface if

  $V \cdot N > 0$



- If the object descriptions have been converted to projection coordinates and our viewing direction is parallel to the viewing Z_v axis, then $V = (0, 0, V_z)$ and

  $V \cdot N = V_z C$.



- As V_z is negative (V's direction is towards negative Z_v axis) we need to consider only C.
- In a right-handed viewing system (viewing direction along negative Z_v axis) the polygon is back face if $C < 0$.
- Also we can't see any face whose normal's z component $C = 0$.
- So in general we can say any polygon as a back face if its normal vector has z-component value

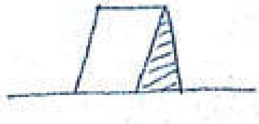
  $C \leq 0$.

- Similarly in a left-handed viewing system the polygon is backface if

  $C \geq 0$

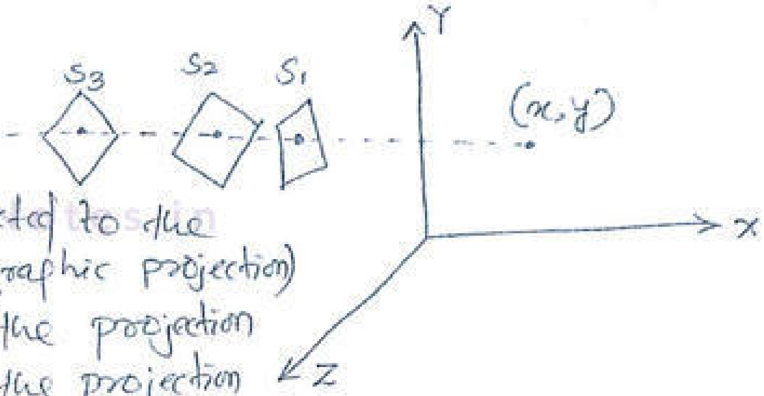
- <u>Limitation:</u>

  It can't be applicable for all situation like overlapping case (if two objects are overlapping) or two faces of a single object is overlapping.



Z-Buffer Method / Depth Buffer Method

- It is a image space method of visible surface detection.
- Here the object depth is usually measured from the view plane along the z-axis of a viewing system.



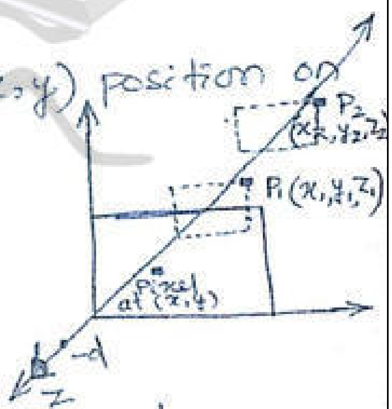
S_1, S_2, S_3 are the surfaces of an object. These surfaces are projected to the viewplane. (orthographic projection)
Let (x, y) is the projection of (x, y, z) on the projection plane
So by measuring the z values of a point we can determine which surface will be visible. As in the diagram surface S_1 is closest to the projection plane, it will be visible.

- Z-Buffer method uses the following two buffers:
  - ✓ Depth buffer
    - Stores depth values for each (x, y) position as surfaces are processed.
    - All positions are initialized to 0 (minimum depth)
  - ✓ Refresh buffer
    - Stores the intensity values for the point (x, y) whose z value is calculated and stored in depth buffer
    - All positions are initialized to the background intensity.
- The z-coordinate to be calculated is normalized one i.e $0 < z < 1$ (When z=0 it is back clipping plane and when z=1 it is front clipping plane)
- The algorithm determines which object is visible at each pixel.

- Basic idea:
  - ✓ Scan convert each polygon surface one at a time
  - ✓ Calculate the depth (z value) at each (x,y) pixel position
  - ✓ The calculated depth is compared to the value previously stored in the depth buffer at that position.
  - ✓ If the calculated depth is smaller than the value stored in the depth buffer, the new depth value is stored and the surface intensity at that position is determined and placed in the same xy location in the refresh buffer.

Algorithm

1. Initialize the depth buffer and refresh buffer so that for all buffer position (x,y),
   $depth(x,y) = 0$, $refresh(x,y) = I_{backgnd}$.
2. For each position on each polygon surface compare depth values to previously stored values in depth buffer to determine visibility
   - Calculate the depth z for each (x,y) position on the polygon.
   - If $z < depth(x,y)$ then set
     $depth(x,y) = z$
     $refresh(x,y) = I_{surf}(x,y)$



where $I_{backgnd}$: value for background intensity
$I_{surf}(x,y)$: projected intensity value for the surface at pixel position (x,y)

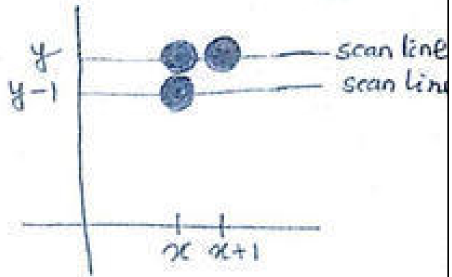
Calculation of depth (calculated from plane equ^n of each surface)

We know the equ^n of the plane is

$$Ax + By + Cz + D = 0$$

$$\Rightarrow z = \frac{-Ax - By - D}{C} \quad \text{———— (1)}$$

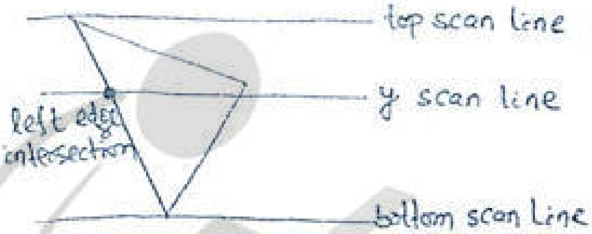
- If the depth of position (x, y) has been determined (let it be z) then the depth z' of the next position $(x+1, y)$ along the scan line is obtained as



$$z' = \frac{-A(x+1) - By - D}{C} \quad \text{——— (2)}$$

$$z' = \frac{-Ax - A - By - D}{C} = \frac{-Ax - By - D}{C} - \frac{A}{C} = z - \frac{A}{C} \quad \text{——— (3)}$$

- On each scan line, we start by calculating the depth on a left edge of the polygon that intersects the scan line. Depth values at each successive position across the scan line are calculated by eqn (3)



- We process the surface from the topmost scan line to the bottom scan line.
- Suppose we move from scan line y to scan line $y-1$. Then we can calculate x positions down a left edge of polygon as $x' = x - \frac{1}{m}$, where m is the slope. Depth values for $y' = y-1$ & $x' = x - 1/m$ is calculated as:

$$z' = \frac{-A\left(x - \frac{1}{m}\right) - B(y-1) - D}{C} \quad \text{——— (4)}$$

$$\Rightarrow z' = \frac{-Ax - By - D}{C} + \frac{A/m + B}{C}$$

$$\Rightarrow z' = z + \frac{A/m + B}{C} \quad \text{——— (5)}$$

- If we process down a vertical edge, the slope is infinite and the recursive calculation reduce t

$$z' = z + \frac{B}{C}$$

Advantages

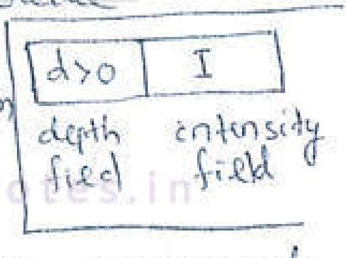
1. Simple as we require only 2 buffers to store the information
2. Popular method used by Microsoft, windows, Linux.
3. Can be implemented in hardware to overcome the speed problem.

Drawback

1. This algorithm is suitable for opaque objects & not suitable for transparent objects
2. Large no. of memory is required as we are using image space approach.

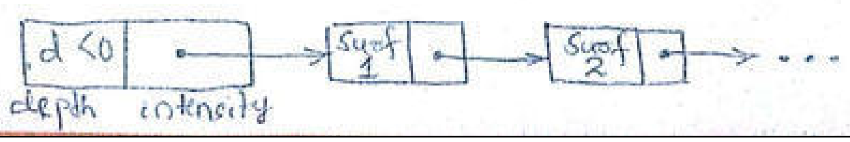
A-Buffer Method

- It is an extension of the ideas in the depth-buffer method.
- This method represents an antialiased, area-averaged, accumulation-buffer method
- It is suitable for both opaque & transparent objects.
- The A-buffer method expands the depth buffer so that each position in the buffer can reference a linked list of surfaces.
- Several intensities can be considered at each pixel position.
- Each position in the A-buffer has two fields:
  - ✓ depth field: stores a positive or negative real no.
  - ✓ intensity field: stores surface-intensity information or a pointer value
- If the depth field is positive, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area.



The I field stores the RGB components of the surface color at that point and the percentage of pixel coverage.

- If the depth field is negative, it indicates multiple surface contributions to the pixel intensity. The intensity field stores a pointers to a linked list of surface data.



Data for each surface in the linked list includes:

- RGB intensity components
- opacity parameters (percent of transparency)
- depth
- percentage of area coverage
- surface identifier
- others surface-rendering parameters
- pointers to the next surface

- The A-buffer can be constructed using methods similar to depth-buffer algorithm
- Using the opacity factors & percentage of surface overlaps, we can calculate the intensity of each pixel as an average of the contributions from the overlapping surfaces.

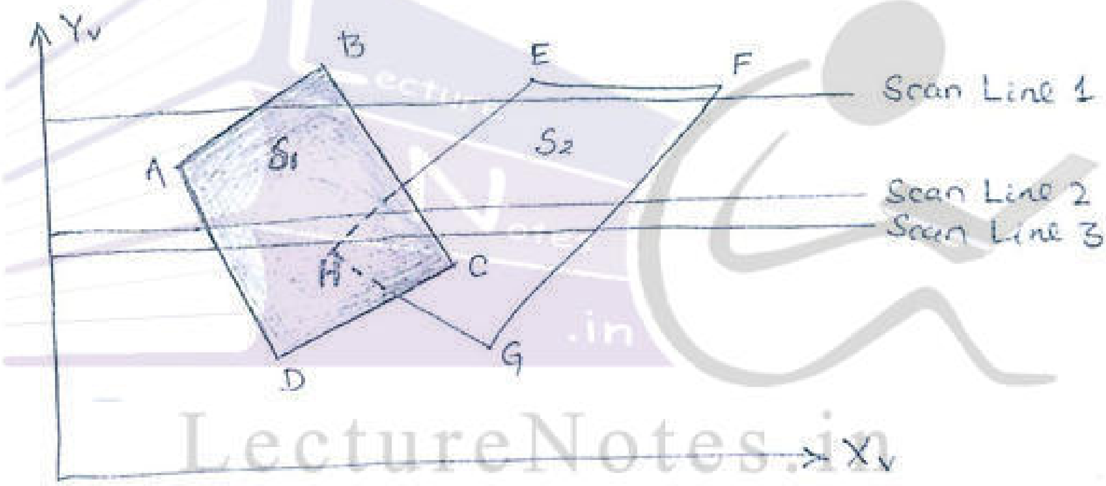
Scan-Line Method

- This is an image-space method for removing hidden surfaces.
- This is an extension of the scan-line algorithm for polygon filling.
- Algorithm deals with multiple surfaces at a time.
- All polygon surfaces intersecting the scan line are examined to determine which are visible.
- Depth calculations are made for each overlapping surface to determine which is nearest to the view plane
- The intensity of the nearest position is entered into the refresh buffer.

- Edge Table
  - ✓ contains coordinate endpoints for each line in the scene
  - ✓ contains inverse slope for each line in the scene
  - ✓ pointers into the polygon table to identify the surfaces bounded by each line

- Polygon Table contains
  - ✓ coefficients of the plane equation for each surface
  - ✓ Intensity & information for the surfaces
  - ✓ pointers into the edge table

- Active List
  - ✓ contains only the edges that cross the current scan line
  - ✓ edges are sorted in order of increasing x.
- Surface Flag
  - ✓ define a flag for each surface that is set on or off to indicate whether a position along a scan line is inside or outside of the surface.
  - ✓ At the leftmost boundary of a surface the surface flag is turned on and at the rightmost boundary of a surface the surface flag is turned off.

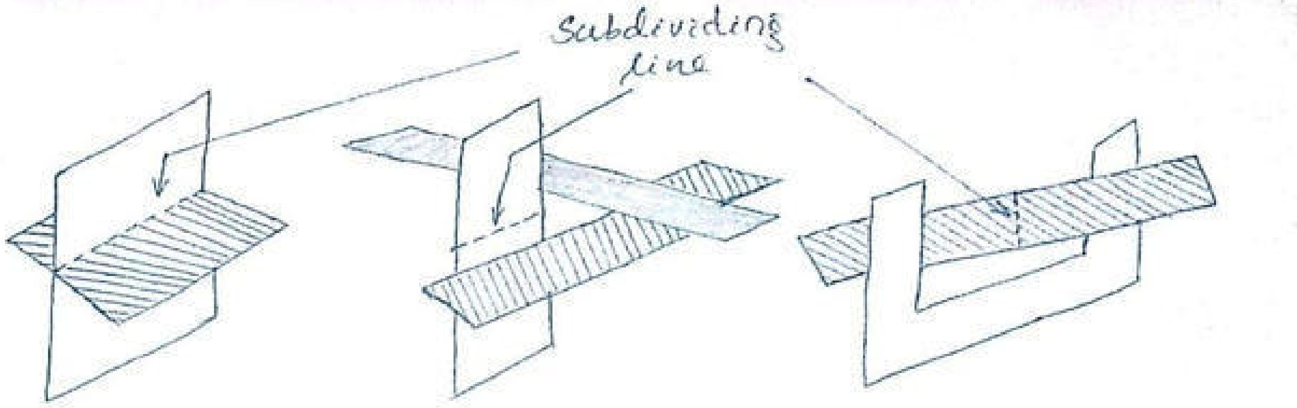


- In the above figure:
  - ✓ Active edge List for scan line 1 : AB, BC, EH, FG
  - ✓ For positions along scan line 1 between edges AB & BC, only the flag for surface S_1 is on. Intensity information for surface S_1 is entered into the refresh buffer.
  - ✓ Between the edges EH & FG, only the flag surface S_2 is on. Intensity of surface S_2 is entered into the refresh buffer
  - ✓ Intensity value in other areas are set to the background intensity.

- For scan line 2
  - ✓ Active edge List : AD, EH, BC, and FG
  - ✓ Between edge AD & EH the flag for S_1 is on.
  - ✓ Between edge BC & FG the flag for S_2 is on
  - ✓ Between edges EH & BC, the flag for both surfaces are on. In this interval depth calculations must be made using the plane coefficients for surface S_1 & S_2.
  - ✓ Assuming depth of S_1 is less than that of S_2, the intensities for surface S_1 are loaded into the refresh buffer until boundary BC is encountered.
  - ✓ After BC the flag for S_1 goes off, and intensities for S_2 are stored until edge FG is passed.

> Coherence : The properties of one part of a scene are related in some way to other parts of a scene are related in some way to other parts of the scene so that the relationship can be used to reduce processing.
> Coherence methods often involve incremental calculations applied along a single scan line or between successive scan lines.

- Advantage of coherence
  - ✓ As we pass from scan line 2 to scan line 3, it has the same active list as of scan line 2.
  - ✓ Since no changes have occurred in line intersections depth calculations between EH & BC is not needed.
- Any no. of overlapping polygon surfaces can be processed with this method.
- When we use coherence we should ensure that surfaces do not cut through or cyclically overlap each other
- If we find this we have to divide the surfaces to eliminate the overlaps.



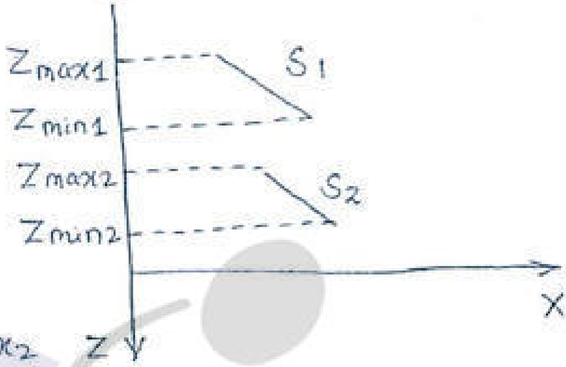
- The above figure shows the overlapping surfaces and where these surfaces could be divided to remove the overlapping

Depth-sorting or Painter's Algorithm

- It uses both image-space and object-space operations.
- It performs the following basic functions:
  1. Surfaces are sorted in order of decreasing depth
  2. Surfaces are scan converted in order, starting with the surface of greatest depth
- Sorting operations are carried out in both image and object space, and the scan conversion of the polygon surfaces is performed in image space.
- The name "painter's algorithm" refers to a simple-minded painter who paints the distant parts of a scene at first and then covers them by those parts that are nearer.

  Painter's algorithm first sort surfaces according to their distance from the view plane. The intensity value for the farthest surface are then entered onto the refresh buffer. Taking each succeeding surface in turn (in decreasing depth order), we paint the surface intensities onto the frame buffer over the intensities of the previously processed surfaces.
- The algorithm paint the polygons in the frame buffer in order of decreasing distance from the viewpoint

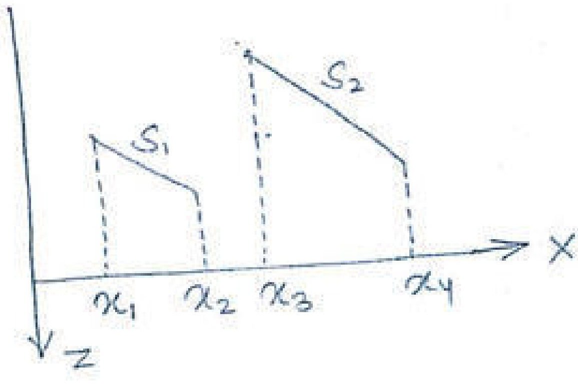
- Broad Steps:
  - ✓ Surfaces are sorted in increasing order of DEPTH.
  - ✓ Resolve ambiguities when polygons overlap (in DEPTH), splitting polygons if necessary.
  - ✓ Surfaces are scan converted in order, starting with the surface of greatest DEPTH.
- Assumption: We are viewing along the $-z$ directions
- Principle:
  - ✓ Each layer of paint (polygon surface of an object) covers up the previous layers while drawing.
- The figure shows two surfaces that overlap in the xy plane but have no depth overlap.
  - ✓ To decide S_1 & S_2 overlaps or not we have to compare z_{min1} with z_{max2}.
  - ✓ In our case $z_{min1} < z_{max2}$ so then are not overlapping.



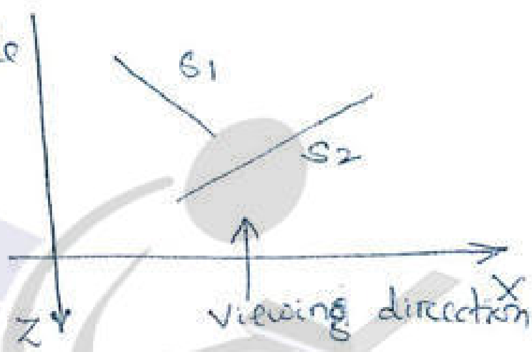
- If depth overlap occurs, additional comparisons are necessary to reorder the surfaces.
- The following set of tests are used to ensure that no re-ordering of surfaces in necessary:
  1. The bounding rectangles in the X-Y plane for the two surfaces do not overlap.
  2. Surface S is completely behind the overlapping surface relative to the viewing position
  3. The overlapping surface is completely in front of S relative to the viewing position
  4. The projections of the two surfaces onto the view plane do not overlap
- We perform these tests in the order listed and proceed to the next overlapping surface as soon as we find one of the tests is true. If all the overlapping surfaces pass at least one of these tests, none of them is behind S so no reordering is then necessary and S is scan converted. Reordering is required if all the four tests fail.

Condition to RE-ORDER the surfaces

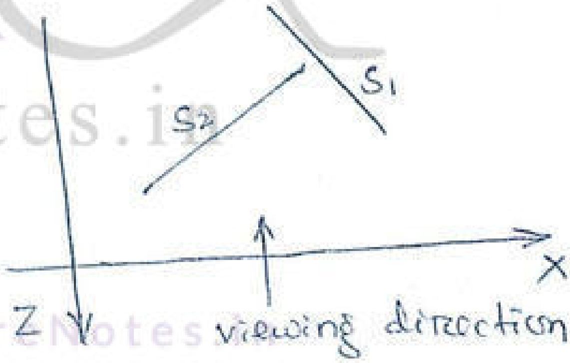
- Checking Test #1:
  - ✓ We first check for overlap in the x direction, then we check for overlap in the y direction. If either of these directions show no overlap, the two planes can't obscure one another.
  - ✓ Here we have depth overlap, but no overlap in x direction.
  - ✓ Hence Test #1 is passed, scan convert S_2 and then S_1.
  - ✓ If the Test #1 fails, then go to Test #2.



- Checking Test #2:
  - ✓ S_1 is competely behind/inside the overlapping surface S_2.
  - ✓ Here surface S is S_1 and the overlapping surface is S_2
  - ✓ Hence Test #2 is passed, scan convert S_1 and then S_2
  - ✓ If the Test #2 fails, then go to Test #3



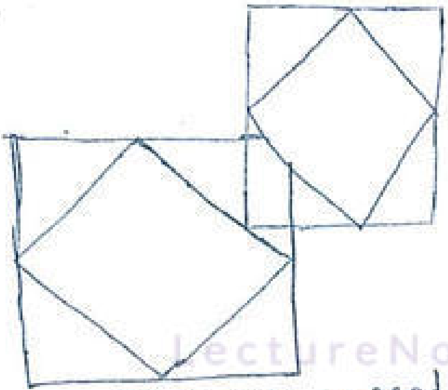
- Checking Test #3:
  - ✓ S_1 is not completely behind S_2. So Test #2 fails
  - ✓ Overlapping surface S_2 is completely front/outside of S_1. So Test #3 passed
  - ✓ So Scan convert S_1 and then S_2.
  - ✓ If the Test #3 fails, then go to Test #4.



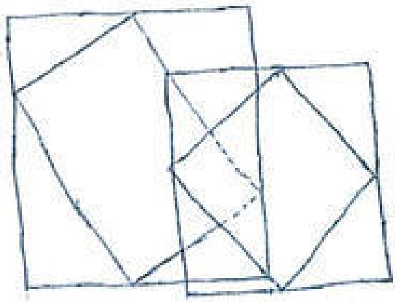
- How to check these conditions (Test #2 and Test #3)?
  - i) Set the plane equation of S_2, such that the surface S_2 is towards the viewing position.
  - ii) Substitute the coordinates of all vertices of S_1 into the plane equation of S_2 and check for the sign
  - iii) If all vertices of S_1 are <u>inside</u> S_2, then S_1 is <u>behind</u> S_2.
  - iv) If all vertices of S_1 are <u>outside</u> S_2, then S_1 is in <u>front</u> of S_2.

- Checking Test #4

To check this test find whether e_i intersects e_j or not where e_i & e_j are the edges of overlapping polygons.



Test #4 passed

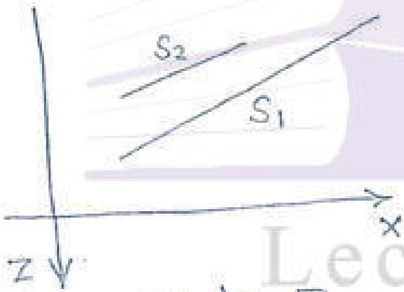


Test #4 fails.

✓ check for the polygons (projections of two surfaces onto the view plane) do not overlap

- If all the tests fail with a particular overlapping surface S', we interchange (reorder) the surfaces S and S' in the sorted order.

Case Study - I



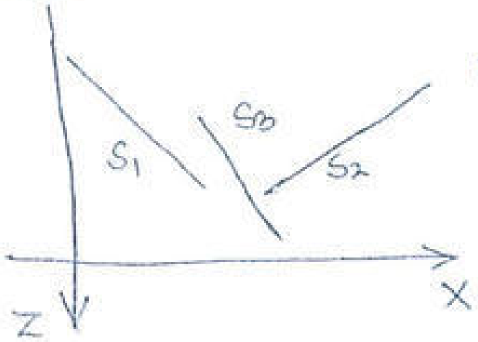
Initial order $S_1 \rightarrow S_2$

All four tests fail

Change the order of surfaces

Final order $S_2 \rightarrow S_1$

Case study II



Initial order $S_1 \rightarrow S_2 \rightarrow S_3$

For Surface S_1 & S_2 Test # 1 passed.

So S_1 & S_2 are correctly ordered.

Now compare S_1 & S_3

Test #1 fails, #2 fails, #3 fails & #4 also fails.

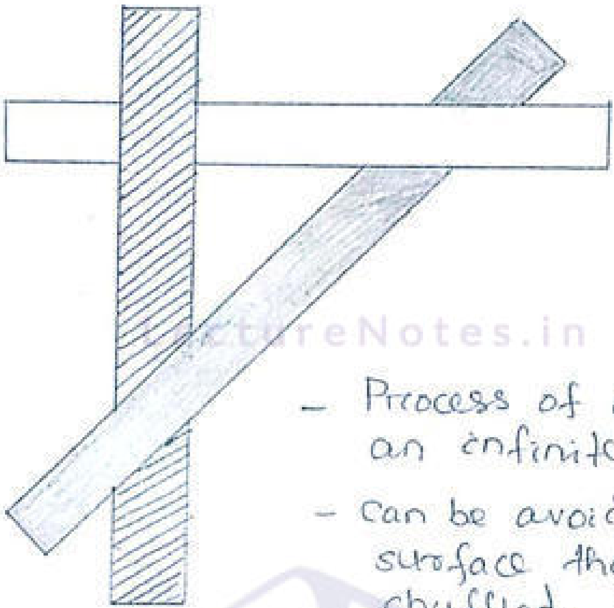
Interchange $S_3 \rightarrow S_2 \rightarrow S_1$

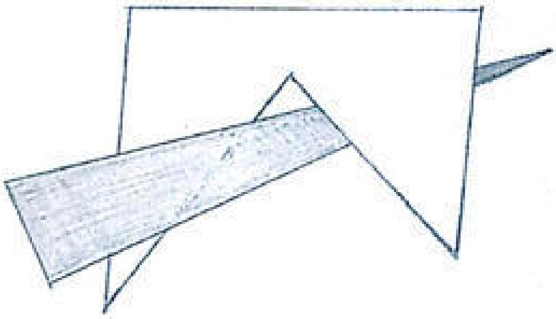
We should not bother for S_2 & S_1 because for any order the test will pass. But we should check for S_3 & S_2

Again all tests fail. So correct order is

$S_2 \rightarrow S_3 \rightarrow S_1$

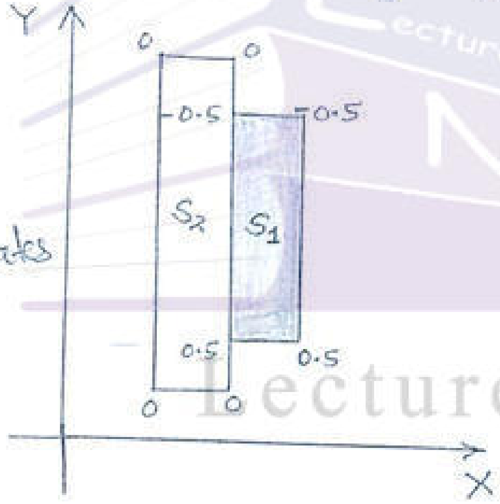
Special Cases

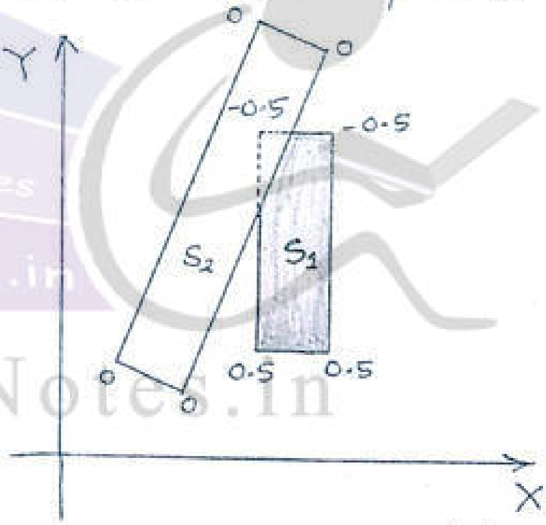




- Process of checking the order results in an infinite loop.
- Can be avoided by setting a FLAG for a surface that has been re-ordered or shuffled. If it has to be altered again, split it into two parts.

values shown are z axis co-ordinates





S_2 is parallel to XY-plane
S_1 is not " " "

Test #1 is passed

Rendering order : $S_1 \rightarrow S_2$

S_2 is parallel to XY-plane but rotated

S_1 is as it is i.e/ not parallel to XY-plane

All tests fail

So we have no other option so split the polygon S_1

Order : Back of $S_1 \rightarrow S_2 \rightarrow$ front of S_1.

Coherence for Visibility

- Coherence is defined as the degree to which parts of an environment or its projection exhibit local similarities such as similarities in depth, color, texture and so on.
- This is important in reducing the computational efforts for object generation and visualization.
- Various kinds of coherence that we can use in visible surface detection algorithms include the following:
  - Object coherence: If one object is entirely separate from another, do not compare
  - Face coherence: Smooth variations across a face; incrementally modify
    (eg: moving from one point to the next on a scan line or moving from one scan line to the next.)
  - Edge Coherence: Visibility changes if a edge crosses behind a visible surface.
  - Implied edge Coherence: Line of intersection of a planar face penetrating another, can be obtained from two points on the intersection.
  - Scanline coherence: Successive lines have similar spans
  - Area Coherence: Span of adjacent group of pixels is often covered by the same visible face.
  - Depth Coherence: Use difference equation to estimate depths of nearby points on the same surface
  - Frame Coherence: Pictures of successive frames of an animation sequence are quite similar (small changes in object and viewpoint)

Computer Graphics

Topic:

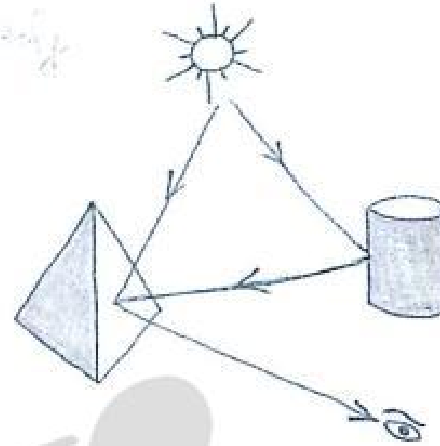
Illumination Models

Contributed By:

Jasaswi Prasad Mohanty

Illumination Models and Surface-Rendering Methods

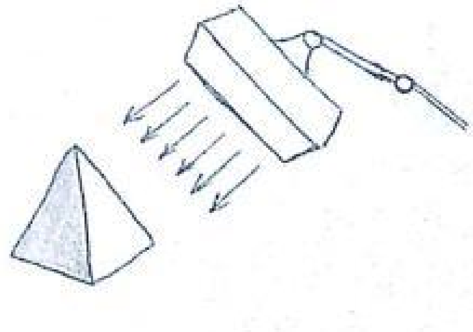
- An illumination model / lighting model / shading model is used to calculate the intensity of light that we should see at a given point on the surface of an object.
- A surface-rendering algorithm uses the intensity calculations from an illumination model to determine the light intensity for all projected pixel positions for the various surfaces in a scene.
- Illumination is a technique to intensify (brightness will be added) an object or point by applying certain model in order to get a realistic image.



Light Sources

- Light source is the source which emits light.
- Total reflected light = Light directly from light source / light emitting source + Light from reflecting surfaces / light reflecting sources
- A surface that is not directly exposed to light may still be visible if nearby objects are illuminated.
- Point source:
  - ✓ Rays from the source follow radially diverging paths from the source.
  - ✓ Dimension is small in comparison to the size of objects in the scene.
  - ✓ Eg: sun.

- Distributed Light Source:
  - ✓ The area of the source is not small compared to the surfaces in the scene
  - ✓ Eg: a long fluorescent light



- Diffuse Reflection:

  - Surfaces that are rough or grainy, tend to scatter the reflected light in all directions. This scattered light is called diffuse reflection.
  - Color of an object is the color of the diffuse reflection of the incident light
    - Eg: 1) A blue object illuminated by a white light source reflects the blue component of the white light and absorbs all other components.
    - 2) The same blue object illuminated by red light appears black since all incident light is absorbed.

- Specular Reflection:

  - Light sources create highlights or bright spots called specular reflection.
  - Mostly found on shiny surfaces.

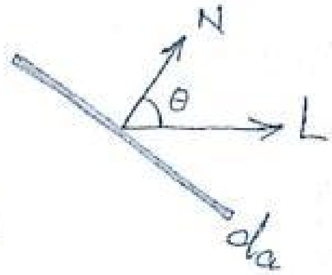
Basic Illumination Models

- Lighting calculations are based on:
  - The optical properties of surfaces
    - opaque or transparent
    - shiny or dull
    - surface - texture
  - The relative position of the surface in a scene
  - The light source specification: color, position
  - the position and orientation of the viewing plane
  - The background lighting conditions

- Ambient light (background light)
  - The light that is the result from the light reflecting off other surfaces in the environment has no direction
  - Each light source has an ambient light contribution, I_a
  - For a given surface, we can specify how much ambient light the surface can reflect using an ambient reflection coefficient, k_a $(0 < k_a < 1)$
  - So the amount of light that the surface reflect is

$$I_{amb} = k_a * I_a$$

- Diffuse Light
  - ✓ The illumination that a surface receives from a light source and reflects equally in all directions
  - ✓ This type of reflection is called Lambertian Reflection (Lambertian surfaces)
  - ✓ The brightness of the surface is independent of the observer position since the light is reflected in all directions equally
- Lambert's Law



  - ✓ How much light the surface receives from a light source depends on the angle between its normal and the vector from the surface point to the light.
  - ✓ Lambert's Law: The radiant energy I_d from a small surface da for a given light source is:

    $$I_d = I_L * \cos\theta$$

    where I_L: the intensity of the light source

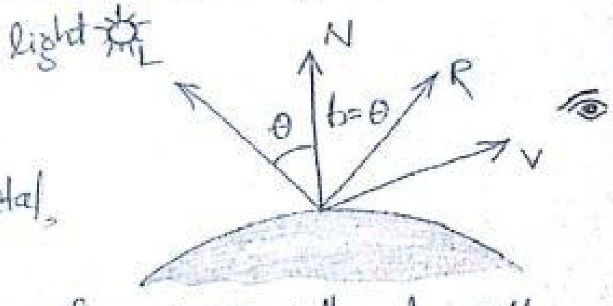
    θ: angle between the surface normal N and light vector L.
- The Diffuse Component
  - ✓ Surface's material property: Assuming that the surface can reflect K_d ($0 < K_d < 1$, diffuse refection coefficient), the amount of diffuse light:

    $$I_{diff} = K_d * I_L * \cos\theta$$

    If N & L are normalized, $\cos\theta = N * L$

    $$\Rightarrow I_{diff} = K_d * I_L * (N * L)$$
  - ✓ The total diffuse reflection = ambient + diffuse

    $$I_{diff} = K_a * I_a + K_d * I_L * (N * L)$$



- Specular Light
  - ✓ These are bright spots on objects (such as polished metal, apple etc.)
  - ✓ Light reflected from the surface unequally to all directions

- ✓ The result of near total reflection of the incident light in a concentrated region around the specular reflection angle.
- ✓ The specular reflection angle equals the angle of the incident light, with the two angles measured on opposite sides of the unit normal surface vector N.
- ✓ Here R : unit vector in the direction of ideal specular reflection
  - L : unit vector directed toward the point light source
  - V : unit vector pointing to the viewer
  - ϕ : viewing angle relative to the specular-reflection direction R.
- ✓ As per Phong specular-reflection model, the intensity of specular reflection is described as:

$$I_{spec} = W(\theta) \cdot I_l \cdot \cos^{n_s}\phi$$

where n_s : specular-reflection parameter
(for a shiny surface $n_s = 100$ or more
for a dull surface $n_s = 1$
for a perfect reflector $n_s = \infty$)

$W(\theta)$: specular-reflection coefficient
At $\theta = 90°$, $W(\theta) = 1$ and all of the incident light is reflected.

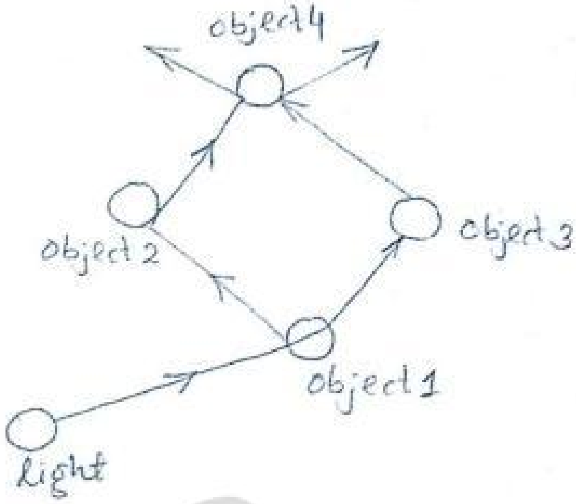
I_l : Intensity of light source

ϕ : viewing angle relative to the specular reflection direction R.

- ✓ Illumination models calculate the intensity projected from a particular surface point in a specified viewing direction
- ✓ Types of illumination model:
  - Global illumination
  - Local illumination

Global Illumination Model

- It takes into account the interaction of light from all the surfaces in the scene.



Local Illumination Model

Only considers the light, the observer position and the object material properties

Illumination Vs Shading

- Illumination (lighting) model determine the color of a surface point by simulating some light attributes
- Shading model applies the illumination models at a set of points and colors the whole image.

Shading Models for Polygons

- constant shading (flat shading): compute illumination at any one point on the surface. Use face or one normal from a pair of edges. Good for far away light & viewers.
- Per-Pixel shading: compute illumination at every point on the surface
- Interpolated shading: compute illumination at vertices and interpolate color.

LECTURENOTES.IN

Computer Graphics

Topic:

Surface Rendering Methods

Contributed By:

Jasaswi Prasad Mohanty

Polygon - Rendering Methods

- The objects are usually polygon-mesh approximations of curved-surface objects.
- Each polygon can be rendered with a single intensity, or the intensity can be obtained at each point of the surface using an interpolation scheme.

157

1. Constant - Intensity Shading / Flat shading

- It is a fast and simple method for rendering an object with polygon surfaces.
- A single intensity is calculated for each polygon. All points over the surface of the polygon are then displayed with the same intensity value.
- Useful for quickly displaying the general appearance of a curved surface
- Flat shading of polygon facets provides an accurate rendering for an object if all of the following assumptions are valid:
  - The object is a polyhedron and is not an approximation of an object with a curved surface.
  - All light sources illuminating the object are sufficiently far from the surface so that N.L and the attenuation function are constant over the surface.
  - The viewing position is sufficiently far from the surface so that V.R is constant over the surface
- Application:
  - This algorithm is applied to the scene where both light source and viewer are far distant from the object.
  - To display fast moving object in a scene, this algorithm is suitable.
- Drawback:
  - Algorithm fails to represent a scene where the intensity is varying uniformly. That means the intensity discontinuities can occur in flat shading.

  Gouraud shading overcomes this limitation.

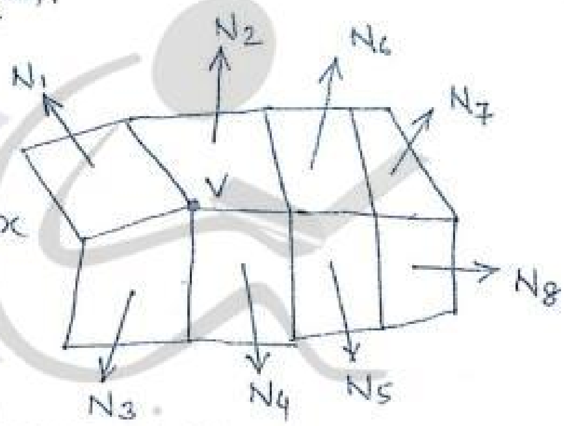
Gouraud Shading

- This is an intensity-interpolation scheme, developed by Gouraud.
- It renders the polygon surface by linearly interpolating intensity values across the surface
- It eliminates the intensity discontinuties (which can occure in flat shading) as intensity values for each polygon are matched with the values of adjacent polygons along the common edges.
- Gouraud shading performs the following calculations:
  - Determine the average unit normal vector at each polygon vertex
  - Apply an illumination model to each vertex to calculate the vertex intensity
  - Linearly interpolate the vertex intensities over the surface of the polygon.

Step 1

✓ At each polygon vertex, we obtain a normal vector by averaging the surface normals of all polygons sharing that vertex

✓ For any vertex position V, we obtain the unit vertex normal



$$N_V = \frac{\sum_{k=1}^{n} N_k}{\left| \sum_{k=1}^{n} N_k \right|}$$

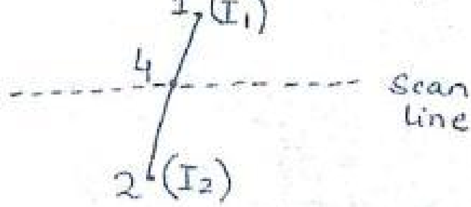
Step 2

✓ After finding the vertex normals at each vertex we can determine the intensity at the vertices from a lighting model.

Step 3

✓ For each scan line, the intensity at the intersection of the scan line with a polygon edge is linearly interpolated from the intensities at the edge end points

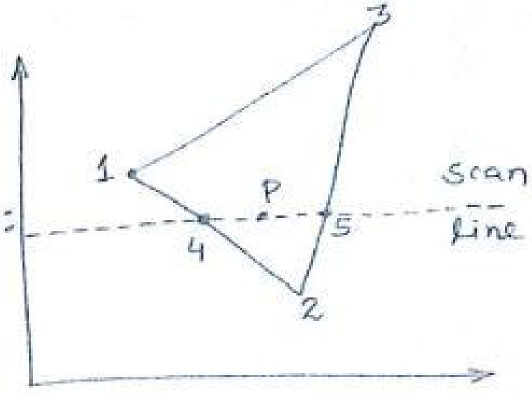
✓ It uses a fast method for obtaining the intensity at point 4 by interpolating between the intensities of point 1 (I_1) & point 2 (I_2) using only the vertical displacement.



Example

Assume that we know the I_1, I_2, I_3 from the step 2.

Intensity of point 4 is calculated as:



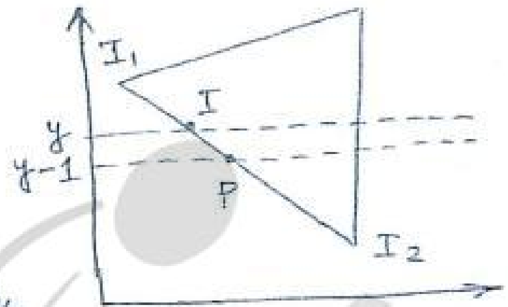
$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2$$

Similarly $I_5 = \frac{y_5 - y_3}{y_2 - y_3} I_2 + \frac{y_2 - y_5}{y_2 - y_3} I_3$

the intensity of an interior point P is calculated as

$$I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5$$

If the intensity at edge position (x, y) is interpolated as



$$I = \frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2$$

then we can obtain the intensity along this edge for the next scan line, $y-1$ as

$$I' = I + \frac{I_2 - I_1}{y_1 - y_2}$$

Advantages

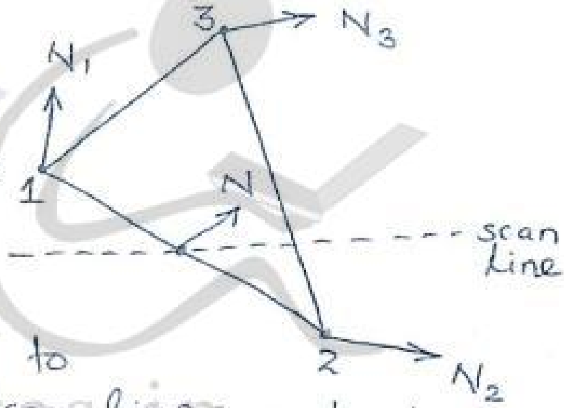
- It can remove the discontinuities associated with the constant-shading model.

Disadvantage

- Highlights on the surface are sometimes displayed with anomalous shapes, and the linear interpolation can cause bright or dark intensity streaks, called Mach bands to appear on the surface. This is overcomed by Phong shading.

Phong Shading / Normal-vector interpolation shading

- this is a more accurate method for rendering polygon surface.
- This method first interpolate normal vectors and then apply the illumination model to each surface point
- steps of Phong Shading
  - ✓ Determine the average unit normal vector at each polygon vertex.
  - ✓ Linearly interpolate the vertex normals over the surface of the polygon
  - ✓ Apply an illumination model along each scan line to calculate projected pixel intensities for the surface points.
- The normal vector N for the scan-line intersection point along the edge between vertices 1 & 2 is obtained as:



$$N = \frac{y - y_2}{y_1 - y_2} N_1 + \frac{y_1 - y}{y_1 - y_2} N_2$$

- ✓ Incremental methods are used to evaluate normals between scan lines and along each individual scan line.
- At each pixel position along a scan line, the illumination model is applied to determine the surface intensity at that point.

Advantages

- Intensity calculations are more accurate than Gouraud shading
- Suitable for shiny surfaces

Disadvantage

- Needs more calculations.

Computer Graphics

Topic:

Computer Animation

Contributed By:

Jasaswi Prasad Mohanty

COMPUTER ANIMATION

1. Introduction

- Computer animation is defined as a technique in which the illusion of movement is created by displaying on a screen or recording on a device, individual states of a dynamic scene.
- The basic idea behind animation is to play back the recorded images at the rates fast enough to fool the human eye into interpreting them as continuous motion. Animation can make a series of dead images come alive. Animation can be used in many areas like entertainment, computer aided-design, scientific visualization, training, education, e-commerce, and computer art.
- It is the time sequence of visual changes in a scene.
- Goal: Synthesize the desired motion effect that involves mixing of natural phenomena, perception and imagination.
- In addition to changing an object position with translations or rotations, an animation can display time variations in an object size, a color, a texture etc.
- Animations can also be generated by changing camera parameters such as position, orientation and focal length.
- Functions:
    - ✓ Storing and managing database
    - ✓ Manipulation and rendering
    - ✓ Camera motion
    - ✓ Generation of intermediate frames

2. Key-frame animation

- A key frame animation consists of an automatic generation of the intermediate frames based on a set of key frames supplied by the animator.
- There are two fundamental approaches to a key frame:
    - ✓ Shape interpolation:
        - ❑ In-betweens (intermediate frames) are obtained by shape interpolation.
        - ❑ Mainly used in film production.
        - ❑ This method transforms one geometrical form into another during an animation.
    - ✓ Parameter interpolation:
        - ❑ Interpolate parameters of the model instead of the object itself.
        - ❑ It produces better image than the first approach.
        - ❑ The parameters are normally spatial parameters, physical parameters, and visualization parameters that decide the model's behavior.

3. Construction of an animation sequence

- A typical animation sequence is obtained using:
    - ✓ **Storyline:** It is a sketch out of the action which defines the motion progression as a set of basic events that must take place. It consists a set of rough sketches, or it could be a list of the basic ideas for the motion.
    - ✓ **Object definition:** An object is a participant in an action that can have some properties and bear relations to other object. An object definition is given to

each participant in the action defined in terms of basic shapes (like circles, polygons etc.)

- ✓ **Key frame specification:** A key frame in an animation is a drawing that defines the starting and ending points of smooth transition. A sequence of key frames defines which movement will be seen, whereas the position of the frames in the animation defines the timing of the movement.
- ✓ **Twining (short form for in-betweening):** It is a process of generating intermediate frames between two images to give an appearance that the first image evolves smoothly into the second image. The "in-betweens" are the drawing between the key frames that help create the illusion of motion. The number of in-between frames needed is determined by the media to be used to display the animation. A film requires 24 frames per second. The time interval for the motion are set up so that there are three to five in-between frames for each pair of key frames.

4. Motion Control Methods

- It is a key issue of computer animation which specifies how an object or an articulated body is animated and may be characterized, according to the type of information to which it is privileged in animating the object or character.
- MCMs may be classified according to the nature of information that is directly manipulated:
  - ✓ Geometric
  - ✓ Physical
  - ✓ Behavioral

4.1 Methods based on geometric and kinematic information:

- ✓ These methods are heavily reliant upon an animator.
- ✓ A motion is locally controlled and defined in terms or coordinates, angles, velocities or accelerations.
- ✓ Different approaches include:
  - o **Performance animation**: It consists of a magnetic or an optical measurement and recording of direct actions of a real person for an immediate or a delayed playback. This technique is used especially in production environments for a 3D character animation.
  - o **Key frame animation**: It is another popular technique in which the animator explicitly specifies the kinematics by supplying the key frame values whose in-between frames are interpolated by the computer.
  - o **Image morphing techniques**: It is a warping-based technique that interpolated the features between two images to obtain a natural in-between image

4.2 Methods based on physical information:

- ✓ In this methods, an animator provides physical data, and the motion is obtained by solving the dynamics equation.
- ✓ Motion is globally controlled.
- ✓ Different methods include:
  - o Parameter adjustment method
  - o Constraint-based method

4.2 Methods based on behavioral information:

- ✓ A behavioral motion control methods drive the behavior of autonomous creatures by providing high-level directives that indicate a specific behavior without any other stimulus.
- ✓ A behavior animation takes into account the relationship among different objects.
- ✓ The control of an animation may be performed at a task level.

5. Procedural animation

- A procedural animation is a type of computer animation, used to automatically generate animation in real time to allow for a more diverse series of actions.
- Procedural animation is used to simulate particle systems (smoke, fire, water etc.), cloth and clothing, rigid-body dynamics, hair and fur dynamics and character animation.
- In computer and video games, it is often used for simple things like turning a character's head when a player looks around.
- Procedural animation corresponds to the creation of a motion by a procedure describing the motion.
- Rules are established for a system, and an initial state is defined for objects in the system. Object locations or parameters for subsequent frames are computed by applying the forces or behaviors defined for the system.
- This type for animation is very useful for generating much life-like motion from relatively little input.
- Here the motion is described by the algorithm or a formula.

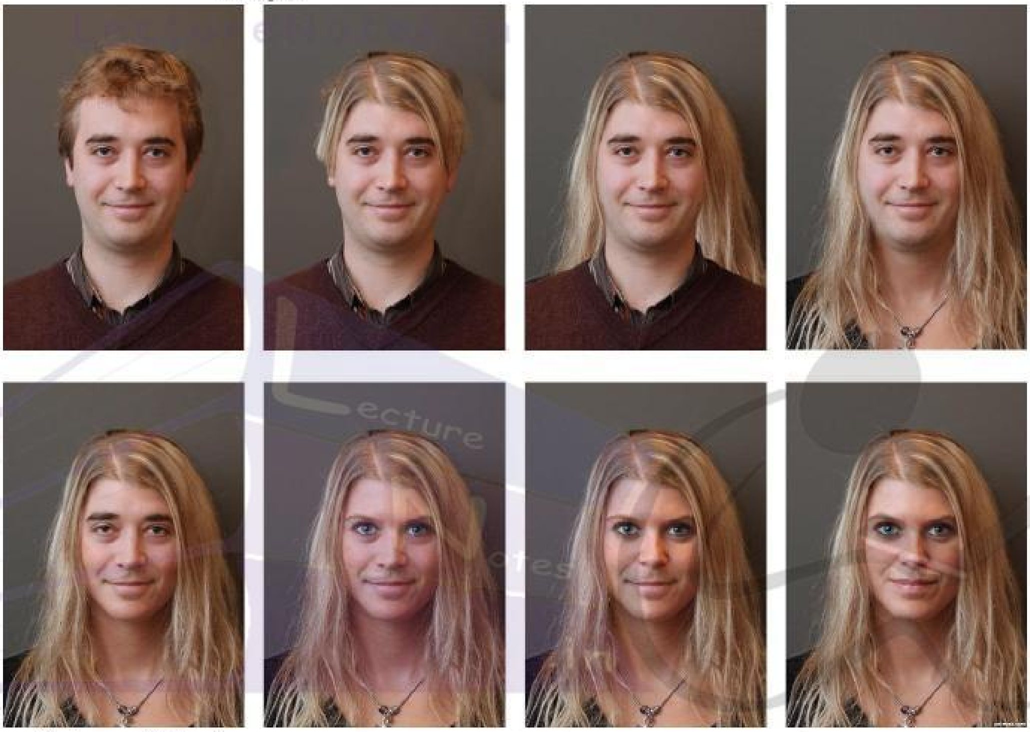
6. Key frame vs. Procedural animation

- To produce a key frame animation:
  - ✓ The animator creates the behavior of a model manually by using an intuitive the "put that there" methodology.
  - ✓ The animator has full and direct control over the positions, shapes and motions of models during the animation.
- To produce a procedural animation:
  - ✓ The animator provides initial conditions and adjusts rather abstract physical parameters such as forces and torques, in order to control positions, shapes and motions of models.
  - ✓ The outcome of varying parameter values is often unpredictable. The animator has to run a simulation to see the result.

7. Introduction to morphing

- Morphing is a phenomenon by which a picture smoothly transmutes into another picture.
- Intermediate image, that bridge the transition are calculated from the source and destination image using a mathematical formula.
- The techniques for calculating intermediate images:
  - ✓ Mesh morphing
  - ✓ Field morphing
- Morphing is a combination of two processes:
  - ✓ Cross-dissolving:
    - ❑ Changes the image's colors pixel by pixel.

- ❑ Produces the bridging images by averaging the pixel colors row-by-row and column-by-column. That is the pixel at row x and column y is the average of the pixel color at (x, y) in the source and the pixel color at (x, y) in the destination image.
- ✓ Warping
  - ❑ Changes the shape of features in an image by shifting its pixel around.
  - ❑ It uses one of the many algorithms to change the row and column values of an image's pixels, thus changing the actual shape of features in an image.



- **Intermediate images**
  - ✓ To make a transition smooth, each intermediate frame is seen as a combination of beginning and ending pictures.
  - ✓ The early images in a sequence are much like the first source image. The middle image of the sequence is the average of the first source image distorted halfway toward the second one and the second source image distorted halfway back toward the first one. The last image in the sequence is similar to the second source image (the destination image).
  - ✓ The i^{th} frame in the sequence is given by

$$(Frame)_i = \left(\frac{i}{Number\ of\ Frames}\right)\%\ Source\ \textbf{blended with}\ \left(\frac{Number\ of\ Frames - i}{Number\ of\ Frames}\right)\%\ Destination$$

  - ✓ This blending of source and destination images produces the much desired gradual smooth transition from source to destination image.
- **Mapping orders**
  - ✓ Image data structures allow storage and access of image in a matrix form either in row major order or in column major order.
  - ✓ A morphing algorithm traverses an image row-by-row, column-by-column or vice versa using a formula to calculate the pixels for a new image.

- ✓ The program can traverse either the source image (forward mapping) or the new image (inverse mapping).
- ✓ Forward mapping iterates over a source image whose pixels already have color values. Any new image starts out blank, with all of its pixels colored white. It visits each pixel in the source image and uses morphing formula to calculate new coordinates for the pixel. Then it paints the source pixel's color in the new image at the calculated set of pixel coordinates.
- ✓ Inverse mapping iterates over the new image. It visits each blank (white) pixel and uses the formula to calculate the coordinates of the pixel in the source image whose color it should copy.

LECTURENOTES.IN

Computer Graphics

Topic:

Virtual Reality Systems

Contributed By:

Jasaswi Prasad Mohanty

VIRTUAL REALITY SYSTEMS

1. Introduction

- Virtual reality (VR) us a technology that allows a user to interact with a computer-simulated environment, be it a real or imaginary one.
- Most current VR environments are primarily visual experiences, displayed either on a computer screen or through special stereoscopic displays.
- Users can interact with a virtual environment or a virtual artifact either through the use of standard input devices or through multimodal devices such as wired glove, boom arm, omni-directional treadmill etc.
- Simulated environment can be similar to the real world (example: simulation for pilot or combat training) or it can differ significantly from reality (example VR games).
- VR is often used to describe a wide variety of applications, commonly associated with its immersive, high visual, 3D environments

2. Design of a VR system

- There is always a trade-off between realism and interactivity.
- The more realistic a scene must appear, the longer it takes to render and the slower the virtual environment will update.
- Detailed images make a virtual environment appear more realistic, but movement through the environment is slow and cumbersome.
- Lesser-detailed scenes will appear false and artificial, but movement through the environment is smooth and faster.
- A VR system consists of six main components: the virtual world, graphics engine, simulation engine, user interface, user inputs, and user outputs.
- A virtual world is a scene database that contains geometric representation and attributes for all objects within the environment.
- The graphics engine is responsible for generating the image that a viewer sees.
- The simulation engine does most of the work required to maintain a virtual environment.
- The user interface controls how the user navigates and interacts with this virtual environment.

3. Important factors in a VR system

a) **Visual realism:** The level of realism in a scene aids considerably in making a believable environment. Ray tracer and professional animation systems produce realistic images used in special effects for movie production.

b) **Image resolution:** Image resolution is another factor that is closely linked with visual realism. Computer-generated images consist of discrete picture elements of pixels, the size and number of these being dependent on the display size and resolution.

c) **Frame rate:** To give the impression of a dynamic picture, the system updates the display very frequently with a new image. In order for a virtual environment to appear flicker free, the system must update the image greater than 20 times each second.

d) **Latency:** It is the most important aspects pf a VR system that must be addressed to make the environment not only more realistic but also tolerable. It is the delay induced by various components of a VR system between a user's input and the corresponding response from the system in the form of a change in the display.

4. **Types of VR system**

   1. **Window-on-world (or desktop) VR:**
      - It is the most common and accessible form of VR system.
      - It involves displaying a 3D virtual world on a regular desktop display without using a specialized movement-tracking equipment.
      - The system do not rely on any specialized input or output devices in order to use them.
      - A user can interact with that environment, but is not immersed in it.
   2. **Video mapping VR:**
      - A video mapping VR uses cameras to project an image of a user into a computer program.
      - Monitoring the user with a video camera provides another form of interactive environment. The computer identifies the user's body and overlays it upon a computer generated scene.
      - By gesturing and moving around in front of the camera, the user can interact with the virtual environment.
   3. **Immersive VR:**
      - An immersive VR uses an head mounted display (HMD) to project a video directly in front of the user's eye, plays audio directly into the user's ear, and tracks whereabouts the user's head.
      - A data glove (or data suit) is used to track movement of the user's body and then duplicates them on the virtual environment.
   4. **Telepresence:**
      - It is a technology that links remote sensors in the real world with the senses of a human operator.
      - It links remote sensors and cameras in the real world with an interface to a human operator.
      - The operator can see the environment that the robot is in and can control its position and actions from a safe distance.
      - Example: use of remote robots in bomb disposal, use of remotely-operated vehicles by fire fighters, use of small instruments on cables by the surgeons etc.
   5. **Augmented reality:**
      - An augmented (or mixed) reality provides a half way point between a non-immersive and fully immersive VR system.
      - Here the computer-generated inputs are merged with the telepresence inputs and the users view of the real world.
      - Example: Head-up displays (HUD) used in modern military aircraft.
   6. **Fish tank VR:**
      - It is used to describe a hybrid system that incorporates a standard desktop VR system with a stereoscopic viewing and head-tracking mechanism.
      - The system uses LCD shutter glasses to provide the stereoscopic images and a head-tracker that monitors the user's point view on the screen.

5. **Advantages of Virtual reality**
   - It gives disabled people the opportunity to join the activities not usually available to them.
   - VR has very important uses in all types of architecture and industrial design.
   - Computer-aided design (CAD) has been an important design tool because it allows the user to draw 3D images on a computer screen.
6. **Input and output devices for Virtual reality**
   a) **Three-dimensional position trackers:** Tracking devices allow a VR system to monitor the position and orientation of selected body parts of a user. In tracking devices, such as HMDs, the position and orientation of the head is measured. A tracking device attached to a glove measures the position and orientation of a hand. Tracking devices, also called 6-degree-of-freedom (6DOF) devices, work by measuring the position (x, y, and z coordinates) and the orientation (yaw, pitch and roll) with respect to a reference point or state. In terms of hardware, three components are in general required: a source that generates a signal, a sensor that receives the signal, and a control box that processes the signal and communicates with the computer. The special purpose hardware used in virtual reality to measure the real-time change in a 3D object position and orientation is called a tracker.
      i) **Mechanical trackers**: It is similar to a robot arm and consists of a jointed structure with rigid links, a supporting base, and an "active end" that is attached to the body part being tracked. These trackers are fast, accurate and not susceptible to jitter. It has a restricted area of operation so tracking of two body parts at the same time is difficult.

(a) (b)

      ii) **Electromagnetic trackers**: Electromagnetic tracker or magnetic tracker allows several body parts to be tracked simultaneously and functions correctly if objects come between the source and detector. In this type of tracker the source produces three electromagnetic fields each of which is perpendicular to the other. The detector on the user's body measures field attenuation (strength and direction of the electromagnetic field) and sends this information back to a computer. These trackers are popular but they are inaccurate. They suffer from latency problems, distortion of data and by large amount of metals. The detector must be within a restricted range from the source, so the user has a limited working volume.

iii) **Ultrasonic trackers**: An ultrasonic tracker is a noncontact position measurement device that uses an ultrasonic signal produced by a stationary transmitter to determine the real-time position of a moving receiver element. There are two ways to calculate position and orientation: phase coherence and time-of-flight. Unlike electromagnetic trackers that are affected by large amounts of metal, ultrasonic trackers do not suffer from this problem. Ultrasonic trackers must have a direct line-of-sight from the emitter to the detector. These trackers are affected by temperature and pressure changes and the humidity level of the work environment.

iv) **Infrared trackers**: IR (optical) trackers are a class of optical tracker, which is a noncontact measurement device that used optical sensing to determine the real-time position or orientation of the object. This type of tracker is not affected by large amount of metal, has a high update rate and low latency. The emitters used here must be directly in the line-of-sight of the camera. Any other sources of IR light, high-intensity light, or other glare affect the correctness of the measurement.

v) **Inertial trackers**: Inertial tracking devices allow the user to move about in a comparatively large working volume as there is no cabling between a computer and a tracker. Inertial trackers are self-contained sensors that measure the rate of change in an object orientation and object translation velocity.

b) **Navigation and manipulation interfaces:** Manipulation tasks involve selecting and moving an object. Users need to be able to manipulate virtual objects which includes rotation also. Navigation tasks has two components. Travel involves moving from the current location to the desired point. Wayfinding refers to finding and setting routes to get to a travel goal within the virtual environment. There are three types if travel tasks: exploration, search and maneuvering. Travel techniques can be classified into the following five categories:

- ✓ Physical movement: user moves through the virtual world
- ✓ Manual viewpoint manipulation: use hand motions to achieve movement
- ✓ Steering: direction specification
- ✓ Target-based travel: destination specification
- ✓ Route planning: path specification

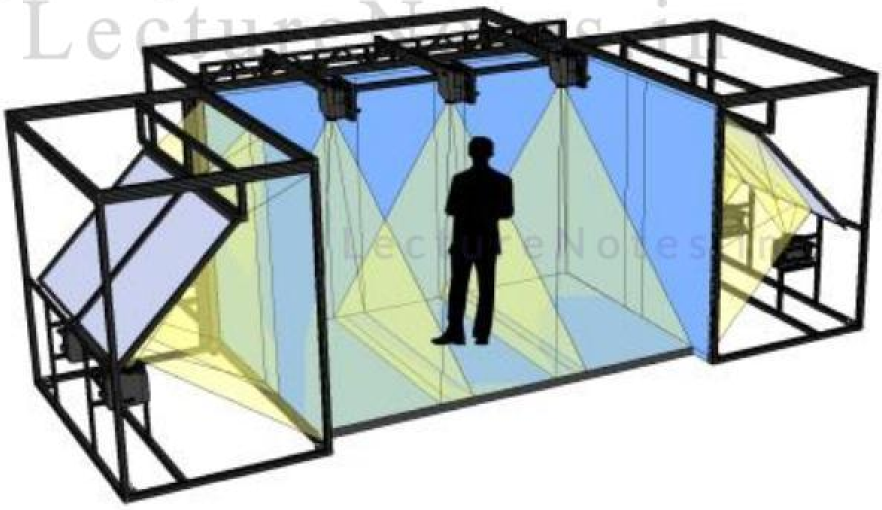
c) **Gesture interfaces:** Navigation and manipulation interfaces limit the freedom of the motion of users to small area or desktop leading to sacrificed and less intuitive virtual world. The solution to this problem is the gesture interfaces. These devices measure the real-time position of the user's fingers or wrist in order to allow natural gesture based interaction with the environment. Gesture recognition is useful for processing information from humans, which is not conveyed through speech. Some of the available gesture device include Fakespace PINCH™ Gloves, 5DT Data Glove and Immersion CyberGlove etc.

d) **Graphics interfaces:** Graphic displays, displays with tracked stereo glasses, glassless displays, multi-projector screen systems and sound display systems are important class of output devices in a virtual environment.

i) **HMD**: It is a computer display we wear on our head. Engineers design HMDs to ensure that no matter in what direction a user might look, the monitor

should stay in front of his eyes. The monitors in an HMD are most often LCD. Any HMDs include speakers or headphones so that it can provide both video and audio output. HMDs almost always include a tracking device so that the point of view displayed in the monitor changes as the user changes his head.

ii) **Cave-automatic virtual environment (CAVE)**: It is a display that uses tracked stereo glasses to feel the environment. It is a small room or cubicle, where at least three walls (and sometime the floor and ceiling) act as giant monitors. The display gives the user a very wide field of view. The user can also move around in a CAVE system without being tethered to a computer. Tracking devices attached to the glasses tell the computer how to adjust the projected images as we walk around the environment.



e) **Sound interface:** Sound effects are often used to communicate important information in video games. This may produce mono, stereo, or 3D audio. Mono sends one signal to every speaker. A stereophonic sound allows for the sounds to seem as if they are coming from anywhere between two speakers. Research into a 3D audio has shown that there are many aspects of our head and ear shape affecting the recognition of 3D sounds. It is possible to apply a rather complex mathematical function called head-related transfer function (HRTF) to a sound to produce this effect.

f) **Examples of input devices:** VR requires a different set of user input tools than traditional computers. There are examples of input devices that have been developed for use with virtual reality.

i) **Glove, DataGlove ad PowerGlove**: A glove device is designed specifically for capturing the movement and location of the hand. When we move our hand, the glove picks up the movement and sends an electrical signal to the computer that translates the movement from the real space into the virtual space.

A DtaGlove is made of lightweight lycra that consists of two measurement tools. The first tool measures the fled and extension of every finger. The second tool measures the absolute position (x, y, and z axes) and orientation (roll, pitch, and yaw) of the hand. This tool has two parts: a stationary transmitter and a receiver, which are placed on the glove. A

PowerGlove is a low-cost version of DataGlove that performs the same function using completely different methods. For flex-measuring, the PowerGlove has a strip of mylar plastic coated with electrically conductive link.

Like DataGlove, the PowerGlove needs calibration for different users. It is less accurate than DataGlove. However, then PowerGlove is more rugged and easier to use than the DataGlove.

ii) **Dexterous hand master**: A dexterous hand master (DHM) is an exoskeleton that is attached to the fingers using Velcro straps, and attached to each finger joint is a device called a half effect sensor, whose purpose is to measure the finger-join angle. DHM uses mechanical linkages to track the movement of the hand. DHM is more accurate than a PowerGlove or a DataGlove. It is also able to measure the side-to-side motion of each finger. Because of its precision it is extremely useful for any application that requires a high level of control such as controlling dexterous robotic hands. DHMs are less sensitive than either DataGlove or a PowerGlove. However, a DHM is rather clunky to work with.

iii) **Mouse and joysticks**: These are sufficient for navigating around a simple virtual world in two dimensions and for performing simple tasks by using the buttons on the devices. Mouse and joysticks usually have two degrees of freedom, although there are mouse designed with six degrees of freedom.

iv) **Wands**: A wand is like a joystick with an unrestrictive base that has 6DOF. There are buttons on a wand and a thumbwheel that allows scalable values to be entered. It can be represented as a drill, paint-brush, spray gun or even an ice-cream cone. A wand is very easy and intuitive to use.

v) **Force (space) balls**: A force ball has a ball on which force is applied. The force we apply is picked up by sensors in the center of the ball from where the information is then relayed to the computer. A force ball has 6DOF. It is easy and intuitive to use. A force ball requires very little space as there is no movement. Most force balls have programmable buttons for a developer to configure to suit the needs of the application. Uses of a force ball are limited to navigation and selection or issuing commands.

vi) **Biological input sensors**: Biosensors are a neural interface technology that detect nerve and muscle activity. Currently, biosensors are used in measuring muscle electrical activity, brain electrical activity, and eye movement. Just as the brain uses the signals to control functions of a human body, these signals can be detected by biosensors and then interpreted by software to control electronics devices, external to the human body.

g) **Haptic feedback:** Haptic technology or haptics is a tactile feedback technology. It makes vibrations and movements which can make people think that there is a real object when there is not. They are used to look into the sense of touch. Some video games use this to make it seem more real. Haptic recreates the sense of touch by applying forces, vibrations, or motions to the user. This mechanical stimulation can be used to assist in the creation of virtual objects in a computer simulation, to control such virtual objects, and to enhance the remote control of machines and devices (tele robotics). Haptic devices may incorporate tactile sensors that measure forces exerted by the user on the interface.