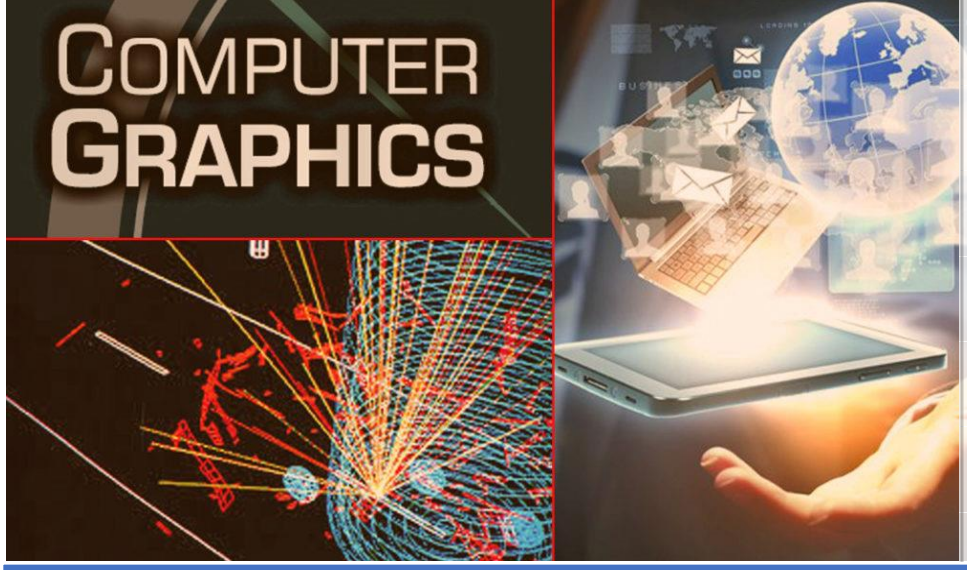


UNIT - I

III YR – I SEM CSE & IT

COMPUTER GRAPHICS

Prepared By: GM SUBHANI, Asst.Professor
MALLAREDDY INSTITUTE OF TECHNOLOGY & SCIENCE
Maisammaguda, Secunderabad, Telangana.

What is computer Graphics?

Computer graphics is an art of drawing pictures, lines, charts, etc. using computers with the help of programming. Computer graphics image is made up of number of pixels. **Pixel** is the smallest addressable graphical unit represented on the computer screen.

Introduction

- Computer is information processing machine. User needs to communicate with computer and the computer graphics is one of the most effective and commonly used ways of communication with the user.
- It displays the information in the form of graphical objects such as pictures, charts, diagram and graphs.
- Graphical objects convey more information in less time and easily understandable formats for example statically graph shown in stock exchange.
- In computer graphics picture or graphics objects are presented as a collection of discrete pixels.
- We can control intensity and color of pixel which decide how picture look like.
- The special procedure determines which pixel will provide the best approximation to the desired picture or graphics object this process is known as **Rasterization.**
- The process of representing continuous picture or graphics object as a collection of discrete pixels is called **Scan Conversion.**

Advantages of computer graphics

- Computer graphics is one of the most effective and commonly used ways of communication with computer.
- It provides tools for producing picture of "real-world" as well as synthetic objects such as mathematical surfaces in 4D and of data that have no inherent geometry such as survey result.
- It has ability to show moving pictures thus possible to produce animations with computer graphics.
- With the use of computer graphics we can control the animation by adjusting the speed, portion of picture in view the amount of detail shown and so on.
- It provides tools called motion dynamics. In which user can move objects as well as observes as per requirement for example walk throw made by builder to show flat interior and surrounding.
- It provides facility called update dynamics. With this we can change the shape color and other properties of object.
- Now in recent development of digital signal processing and audio synthesis chip the interactive graphics can now provide audio feedback along with the graphical feed backs.

Application of computer graphics

- User interface: - Visual object which we observe on screen which communicates with user is one of the most useful applications of the computer graphics.
- Plotting of graphics and chart in industry, business, government and educational organizations drawing like bars, pie-charts, histogram's are very useful for quick and good decision making.
- Office automation and desktop publishing: - It is used for creation and dissemination of information. It is used in in-house creation and printing of documents which contains text, tables, graphs and other forms of drawn or scanned images or picture.

- Simulation and animation: - Use of graphics in simulation makes mathematic models and mechanical systems more realistic and easy to study.
- Art and commerce: - There are many tools provided by graphics which allows used to make their picture animated and attracted which are used in advertising.
- Process control: - Now a day's automation is used which is graphically displayed on the screen.
- Cartography: - Computer graphics is also used to represent geographic maps, weather maps, oceanographic charts etc.
- Education and training: - Computer graphics can be used to generate models of physical, financial and economic systems. These models can be used as educational aids.
- Image processing: - It is used to process image by changing property of the image.

Display devices

- Display devices are also known as output devices.
- Most commonly used output device in a graphics system is a video monitor.

Cathode-ray-tubes

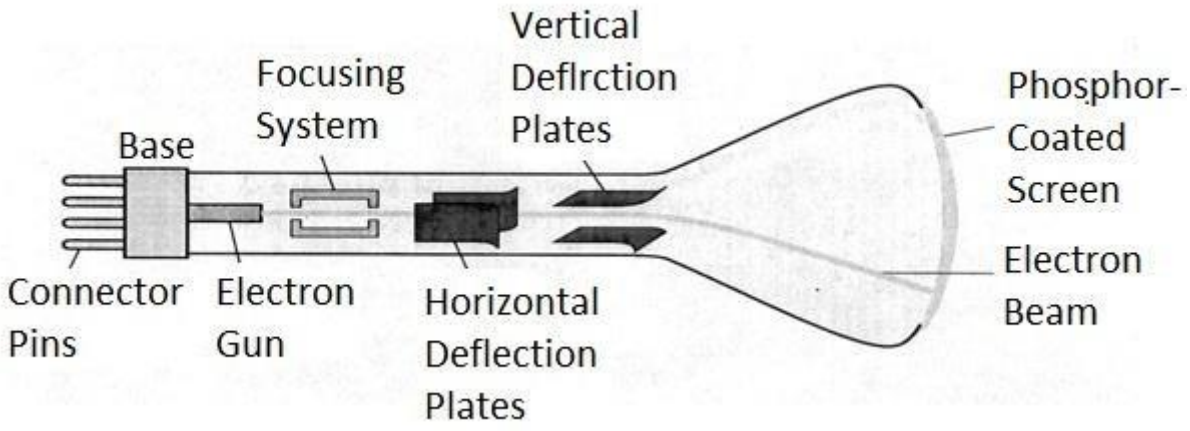


Fig. 1.1: - Cathode ray tube.

- It is an evacuated glass tube.
- An electron gun at the rear of the tube produce a beam of electrons which is directed towards the screen of the tube by a high voltage typically 15000 to 20000 volts
- Inner side screen is coated with phosphor substance which gives light when it is stroked bye electrons.
- Control grid controls velocity of electrons before they hit the phosphor.
- The control grid voltage determines how many electrons are actually in the electron beam. The negative the control voltage is the fewer the electrons that pass through the grid.
- Thus control grid controls Intensity of the spot where beam strikes the screen.
- The focusing system concentrates the electron beam so it converges to small point when hits the phosphor coating.
- Deflection system directs beam which decides the point where beam strikes the screen.
- Deflection system of the CRT consists of two pairs of parallel plates which are vertical and horizontal deflection plates.
- Voltage applied to vertical and horizontal deflection plates is control vertical and horizontal deflection respectively.

- There are two techniques used for producing images on the CRT screen:
    1. Vector scan/Random scan display.
    2. Raster scan display.

Vector scan/Random scan display

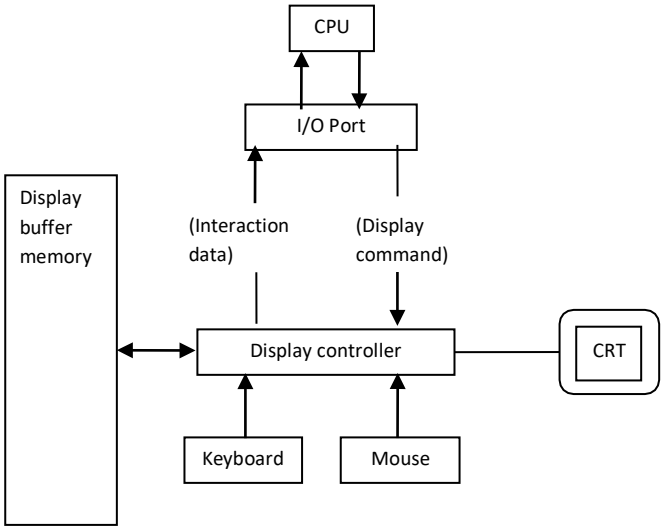


Fig. 1.2: - Architecture of a vector display.

- Vector scan display directly traces out only the desired lines on CRT.
- If we want line between point p1 & p2 then we directly drive the beam deflection circuitry which focus beam directly from point p1 to p2.
- If we do not want to display line from p1 to p2 and just move then we can blank the beam as we move it.
- To move the beam across the CRT, the information about both magnitude and direction is required. This information is generated with the help of vector graphics generator.
- Fig. 1.2 shows architecture of vector display. It consists of display controller, CPU, display buffer memory and CRT.
- Display controller is connected as an I/O peripheral to the CPU.
- Display buffer stores computer produced display list or display program.
- The Program contains point & line plotting commands with end point co-ordinates as well as character plotting commands.
- Display controller interprets command and sends digital and point co-ordinates to a vector generator.
- Vector generator then converts the digital co-ordinate value to analog voltages for beam deflection circuits that displace an electron beam which points on the CRT's screen.
- In this technique beam is deflected from end point to end point hence this techniques is also called random scan.
- We know as beam strikes phosphors coated screen it emits light but that light decays after few milliseconds and therefore it is necessary to repeat through the display list to refresh the screen at least 30 times per second to avoid flicker.
- As display buffer is used to store display list and used to refreshing, it is also called refresh buffer.

Raster scan display

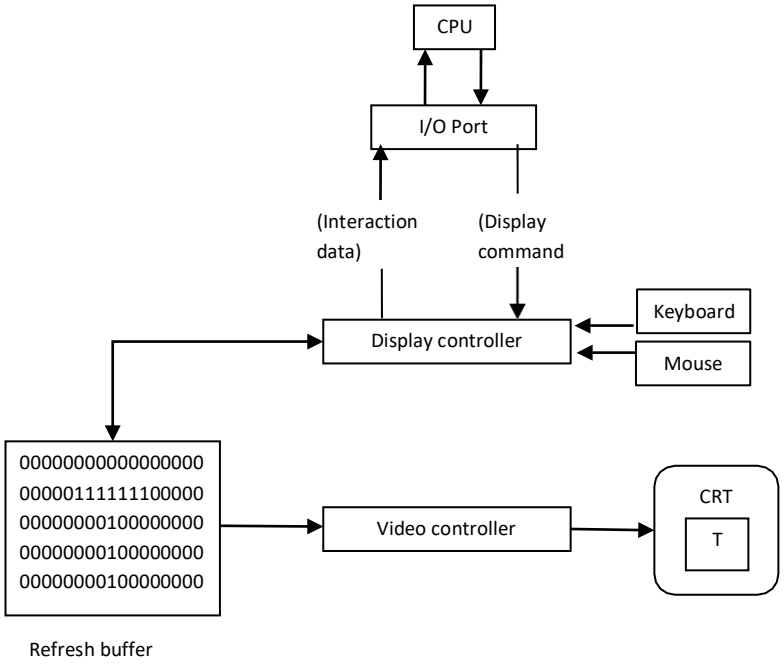


Fig. 1.3: - Architecture of a raster display.

- Fig. 1.3 shows the architecture of Raster display. It consists of display controller, CPU, video controller, refresh buffer, keyboard, mouse and CRT.
- The display image is stored in the form of 1's and 0's in the refresh buffer.
- The video controller reads this refresh buffer and produces the actual image on screen.
- It will scan one line at a time from top to bottom & then back to the top.

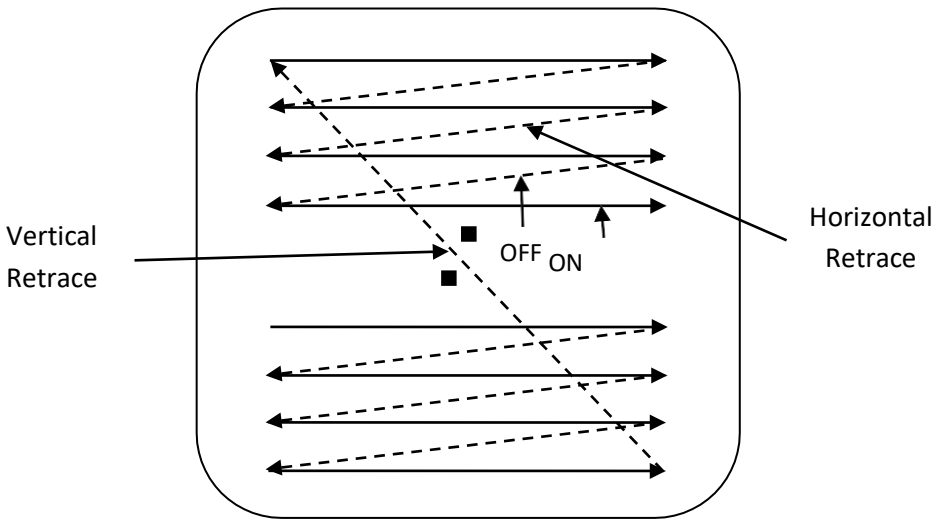


Fig. 1.4: - Raster scan CRT.

- In this method the horizontal and vertical deflection signals are generated to move the beam all over the screen in a pattern shown in fig. 1.4.
- Here beam is swept back & forth from left to the right.
- When beam is moved from left to right it is ON.

- When beam is moved from right to left it is OFF and process of moving beam from right to left after completion of row is known as **Horizontal Retrace.**
- When beam is reach at the bottom of the screen. It is made OFF and rapidly retraced back to the top left to start again and process of moving back to top is known as **Vertical Retrace**.
- The screen image is maintained by repeatedly scanning the same image. This process is known as **Refreshing of Screen.**
- In raster scan displays a special area of memory is dedicated to graphics only. This memory is called **Frame Buffer.**
- Frame buffer holds set of intensity values for all the screen points.
- That intensity is retrieved from frame buffer and display on screen one row at a time.
- Each screen point referred as pixel or **Pel (Picture Element).**
- Each pixel can be specified by its row and column numbers.
- It can be simply black and white system or color system.
- In simple black and white system each pixel is either ON or OFF, so only one bit per pixel is needed.
- Additional bits are required when color and intensity variations can be displayed up to 24-bits per pixel are included in high quality display systems.
- On a black and white system with one bit per pixel the frame buffer is commonly called a **Bitmap.** And for systems with multiple bits per pixel, the frame buffer is often referred as a **Pixmap.**

Difference between random scan and raster scan

Base of Difference	Raster Scan System	Random Scan System
Electron Beam	The electron beam is swept across the screen, one row at a time, from top to bottom.	The electron beam is directed only to the parts of screen where a picture is to be drawn.
Resolution	Its resolution is poor because raster system in contrast produces zigzag lines that are plotted as discrete point sets.	Its resolution is good because this system produces smooth lines drawings because CRT beam directly follows the line path.
Picture Definition	Picture definition is stored as a set of intensity values for all screen points, called pixels in a refresh buffer area.	Picture definition is stored as a set of line drawing instructions in a display file.
Realistic Display	The capability of this system to store intensity values for pixel makes it well suited for the realistic display of scenes contain shadow and color pattern.	These systems are designed for line-drawing and can't display realistic shaded scenes.
Draw an Image	Screen points/pixels are used to draw an image.	Mathematical functions are used to draw an image.

Color CRT monitors

- A CRT monitors displays color pictures by using a combination of phosphors that emit different colored light.
- It produces range of colors by combining the light emitted by different phosphors.
- There are two basic techniques for color display:
    1. Beam-penetration technique
    2. Shadow-mask technique

Beam-penetration technique

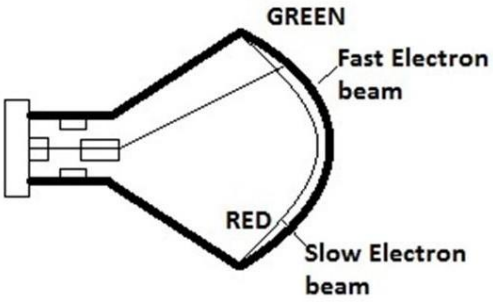


Fig. 1.5: - Beam-penetration CRT

- This technique is used with random scan monitors.
- In this technique inside of CRT coated with two phosphor layers usually red and green. The outer layer of red and inner layer of green phosphor.
- The color depends on how far the electron beam penetrates into the phosphor layer.
- A beam of fast electron penetrates more and excites inner green layer while slow electron excites outer red layer.
- At intermediate beam speed we can produce combination of red and green lights which emit additional two colors orange and yellow.
- The beam acceleration voltage controls the speed of the electrons and hence color of pixel.
- It is a low cost technique to produce color in random scan monitors.
- It can display only four colors.
- Quality of picture is not good compared to other techniques.

Shadow-mask technique

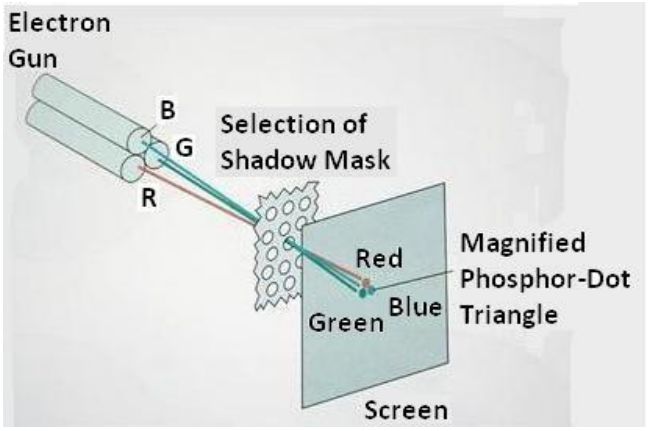


Fig. 1.6: - Shadow-mask CRT.

- It produces wide range of colors as compared to beam-penetration technique.
- This technique is generally used in raster scan displays. Including color TV.
- In this technique CRT has three phosphor color dots at each pixel position. One dot for red, one for green and one for blue light. This is commonly known as **Dot Triangle.**
- Here in CRT there are three electron guns present, one for each color dot. And a shadow mask grid just behind the phosphor coated screen.
- The shadow mask grid consists of series of holes aligned with the phosphor dot pattern.
- Three electron beams are deflected and focused as a group onto the shadow mask and when they pass through a hole they excite a dot triangle.
- In dot triangle three phosphor dots are arranged so that each electron beam can activate only its corresponding color dot when it passes through the shadow mask.
- A dot triangle when activated appears as a small dot on the screen which has color of combination of three small dots in the dot triangle.
- By changing the intensity of the three electron beams we can obtain different colors in the shadow mask CRT.

Direct-view storage tubes (DVST)

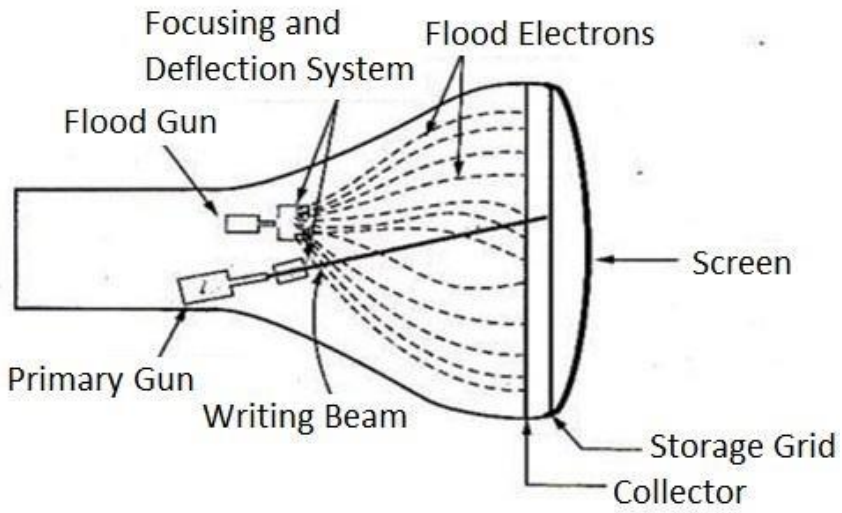


Fig. 1.7: - Direct-view storage tube.

- In raster scan display we do refreshing of the screen to maintain a screen image.
- DVST gives alternative method for maintaining the screen image.
- DVST uses the storage grid which stores the picture information as a charge distribution just behind the phosphor coated screen.
- DVST consists two electron guns a primary gun and a flood gun.
- A primary gun stores the picture pattern and the flood gun maintains the picture display.
- A primary gun emits high speed electrons which strike on the storage grid to draw the picture pattern.
- As electron beam strikes on the storage grid with high speed, it knocks out electrons from the storage grid keeping the net positive charge.
- The knocked out electrons are attracted towards the collector.
- The net positive charge on the storage grid is nothing but the picture pattern.
- The continuous low speed electrons from flood gun pass through the control grid and are attracted to the positive charged area of the storage grid.

- The low speed electrons then penetrate the storage grid and strike the phosphor coating without affecting the positive charge pattern on the storage grid.
- During this process the collector just behind the storage grid smooth out the flow of flood electrons.

Advantage of DVST

- Refreshing of CRT is not required.
- Very complex pictures can be displayed at very high resolution without flicker.
- Flat screen.

Disadvantage of DVST

- They do not display color and are available with single level of line intensity.
- For erasing it is necessary to removal of charge on the storage grid so erasing and redrawing process take several second.
- Erasing selective part of the screen cannot be possible.
- Cannot used for dynamic graphics application as on erasing it produce unpleasant flash over entire screen.
- It has poor contrast as a result of the comparatively low accelerating potential applied to the flood electrons.
- The performance of DVST is somewhat inferior to the refresh CRT.

Flat Panel Display

- The term flat panel display refers to a class of video device that have reduced volume, weight & power requirement compared to a CRT.
- As flat panel display is thinner than CRTs, we can hang them on walls or wear on our wrists.
- Since we can even write on some flat panel displays they will soon be available as pocket notepads.
- We can separate flat panel display in two categories:
    1. **Emissive displays**: - the emissive display or **emitters** are devices that convert electrical energy into light. For Ex. Plasma panel, thin film electroluminescent displays and light emitting diodes.
    2. **Non emissive displays**: - non emissive display or **non emitters** use optical effects to convert sunlight or light from some other source into graphics patterns. For Ex. LCD (Liquid Crystal Display).

Plasma Panels displays

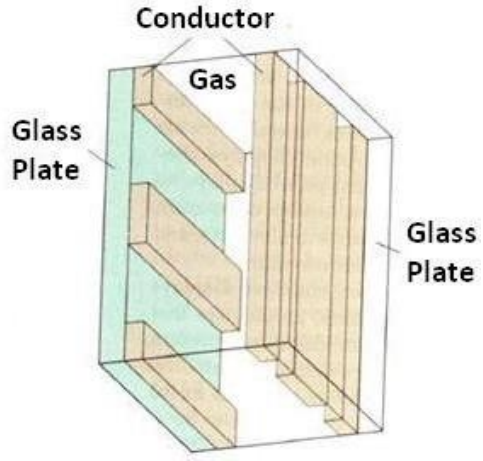


Fig. 1.8: - Basic design of a plasma-panel display device.

- This is also called gas discharge displays.
- It is constructed by filling the region between two glass plates with a mixture of gases that usually includes neon.
- A series of vertical conducting ribbons is placed on one glass panel and a set of horizontal ribbon is built into the other glass panel.
- Firing voltage is applied to a pair of horizontal and vertical conductors cause the gas at the intersection of the two conductors to break down into glowing plasma of electrons and ions.
- Picture definition is stored in a refresh buffer and the firing voltages are applied to refresh the pixel positions, 60 times per second.
- Alternating current methods are used to provide faster application of firing voltages and thus brighter displays.
- Separation between pixels is provided by the electric field of conductor.
- One disadvantage of plasma panels is they were strictly monochromatic device that means shows only one color other than black like black and white.

Thin Film Electroluminescent Displays.

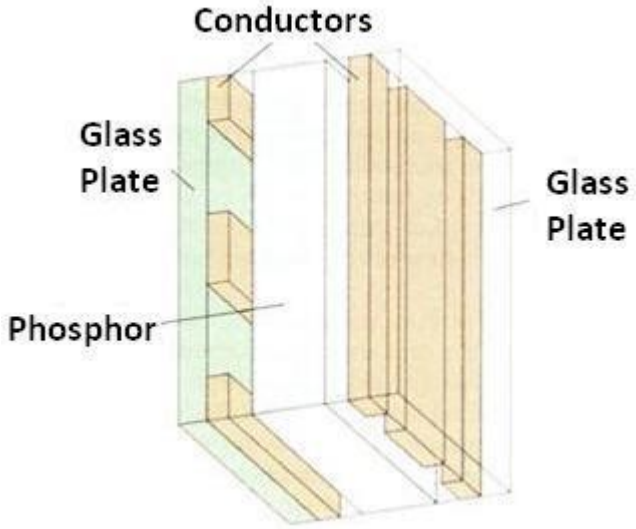


Fig. 1.9: - Basic design of a thin-film electro luminescent display device.

- It is similar to plasma panel display but region between the glass plates is filled with phosphors such as zinksulphide doped with magnesium instead of gas.
- When sufficient voltage is applied the phosphors becomes a conductor in area of intersection of the two electrodes.
- Electrical energy is then absorbed by the manganese atoms which then release the energy as a spot of light similar to the glowing plasma effect in plasma panel.
- It requires more power than plasma panel.
- In this good color and gray scale difficult to achieve.

Light Emitting Diode (LED)

- In this display a matrix of multi-color light emitting diode is arranged to form the pixel position in the display. And the picture definition is stored in refresh buffer.
- Similar to scan line refreshing of CRT information is read from the refresh buffer and converted to voltage levels that are applied to the diodes to produce the light pattern on the display.

Liquid Crystal Display (LCD)

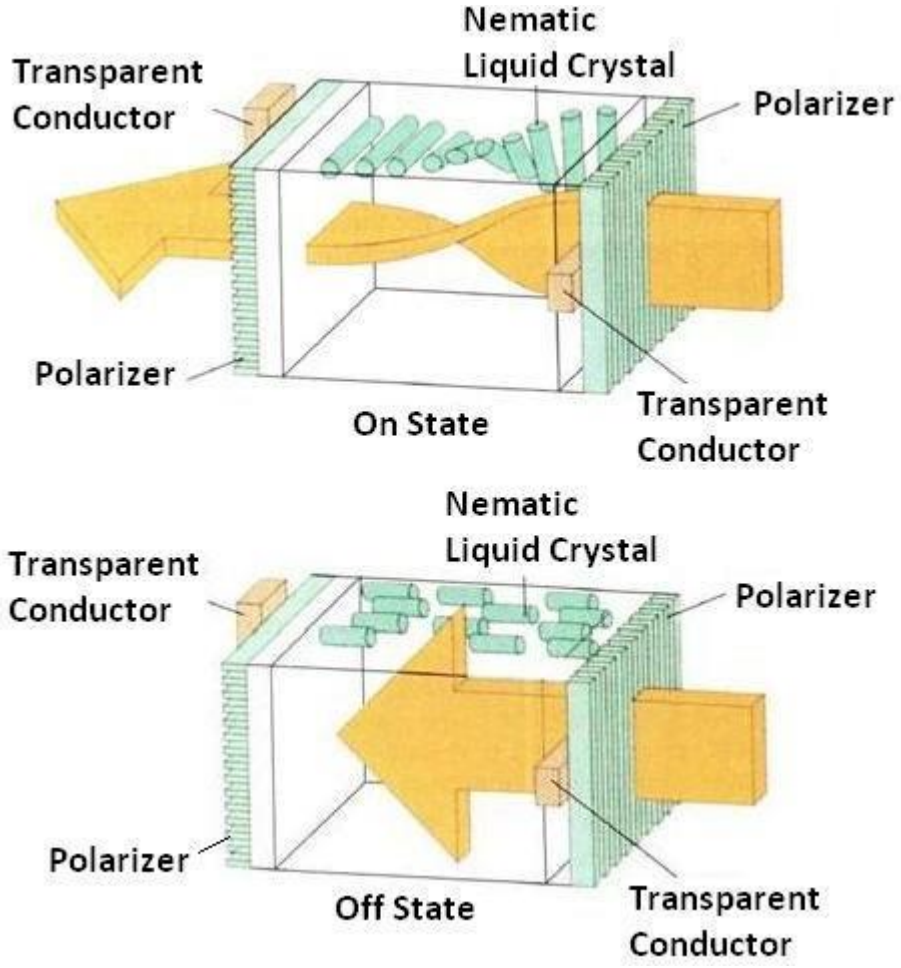


Fig. 1.10: - Light twisting shutter effect used in design of most LCD.

- It is generally used in small system such as calculator and portable laptop.
- This non emissive device produce picture by passing polarized light from the surrounding or from an internal light source through liquid crystal material that can be aligned to either block or transmit the light.
- The liquid crystal refreshes to fact that these compounds have crystalline arrangement of molecules then also flows like liquid.
- It consists of two glass plates each with light polarizer at right angles to each other sandwich the liquid crystal material between the plates.
- Rows of horizontal transparent conductors are built into one glass plate, and column of vertical conductors are put into the other plates.
- The intersection of two conductors defines a pixel position.
- In the ON state polarized light passing through material is twisted so that it will pass through the opposite polarizer.
- In the OFF state it will reflect back towards source.
- We applied a voltage to the two intersecting conductor to align the molecules so that the light is not twisted.
- This type of flat panel device is referred to as a passive matrix LCD.
- In active matrix LCD transistors are used at each (x, y) grid point.

- Transistor cause crystal to change their state quickly and also to control degree to which the state has been changed.
- Transistor can also serve as a memory for the state until it is changed.
- So transistor make cell ON for all time giving brighter display then it would be if it had to be refresh periodically

Advantages of LCD display

- Low cost.
- Low weight.
- Small size
- Low power consumption.

Three dimensional viewing devices

- The graphics monitor which are display three dimensional scenes are devised using a technique that reflects a CRT image from a vibrating flexible mirror.

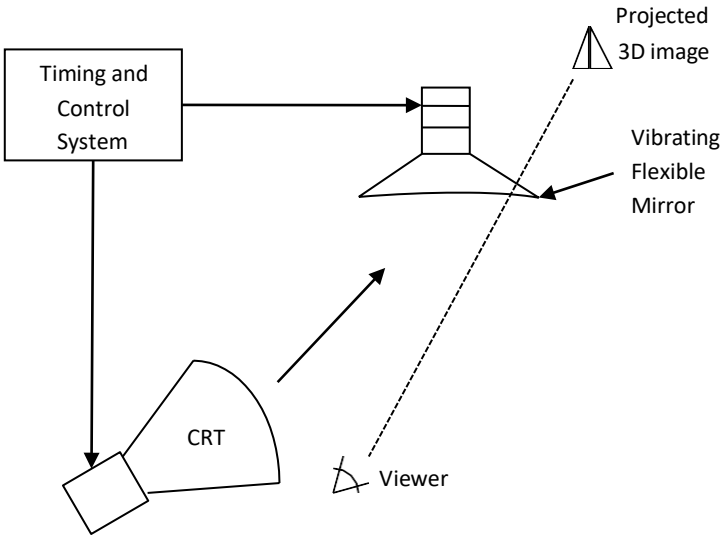


Fig. 1.11: - 3D display system uses a vibrating mirror.

- Vibrating mirror changes its focal length due to vibration which is synchronized with the display of an object on CRT.
- The each point on the object is reflected from the mirror into spatial position corresponding to distance of that point from a viewing position.
- Very good example of this system is GENISCO SPACE GRAPH system, which use vibrating mirror to project 3D objects into a 25 cm by 25 cm by 25 cm volume. This system is also capable to show 2D cross section at different depth.

Application of 3D viewing devices

- In medical to analyze data from ultra-sonography.
- In geological to analyze topological and seismic data.
- In designing like solid objects viewing and 3D viewing of objects.

Stereoscopic and virtual-reality systems

Stereoscopic system

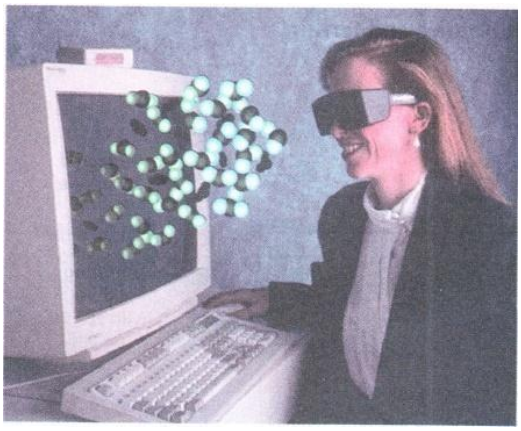


Fig. 1.12: - stereoscopic views.

- Stereoscopic views does not produce three dimensional images, but it produce 3D effects by presenting different view to each eye of an observer so that it appears to have depth.
- To obtain this we first need to obtain two views of object generated from viewing direction corresponding to each eye.
- We can construct the two views as computer generated scenes with different viewing positions or we can use stereo camera pair to photograph some object or scene.
- When we see simultaneously both the view as left view with left eye and right view with right eye then two views is merge and produce image which appears to have depth.
- One way to produce stereoscopic effect is to display each of the two views with raster system on alternate refresh cycles.
- The screen is viewed through glasses with each lance design such a way that it act as a rapidly alternating shutter that is synchronized to block out one of the views.

Virtual-reality

Fig. 1.13: - virtual reality.

- Virtual reality is the system which produce images in such a way that we feel that our surrounding is what we are set in display devices but in actually it does not.
- In virtual reality user can step into a scene and interact with the environment.

- A head set containing an optical system to generate the stereoscopic views is commonly used in conjunction with interactive input devices to locate and manipulate objects in the scene.
- Sensor in the head set keeps track of the viewer's position so that the front and back of objects can be seen as the viewer "walks through" and interacts with the display.
- Virtual reality can also be produce with stereoscopic glass and video monitor instead of head set. This provides low cost virtual reality system.
- Sensor on display screen track head position and accordingly adjust image depth.

Raster graphics systems

Simple raster graphics system

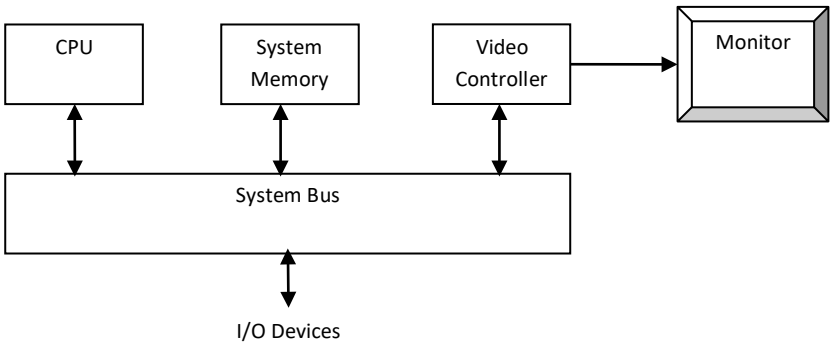


Fig. 1.14: - Architecture of a simple raster graphics system.

- Raster graphics systems having additional processing unit like video controller or display controller.
- Here frame buffer can be anywhere in the system memory and video controller access this for refresh the screen.
- In addition to video controller more processors are used as co-processors to accelerate the system in sophisticated raster system.

Raster graphics system with a fixed portion of the system memory reserved for the frame buffer

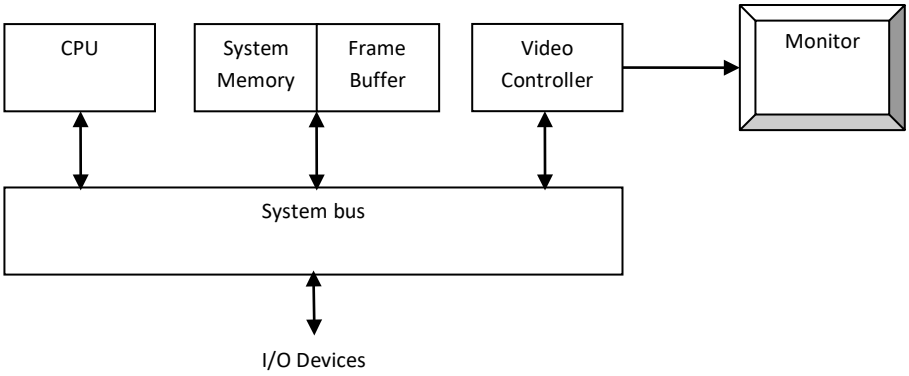


Fig. 1.15: - Architecture of a raster graphics system with a fixed portion of the system memory reserved for the frame buffer.

- A fixed area of the system memory is reserved for the frame buffer and the video controller can directly access that frame buffer memory.
- Frame buffer location and the screen position are referred in Cartesian coordinates.
- For many graphics monitors the coordinate origin is defined at the lower left screen corner.
- Screen surface is then represented as the first quadrant of the two dimensional systems with positive X-value increases as left to right and positive Y-value increases bottom to top.

Basic refresh operation of video controller

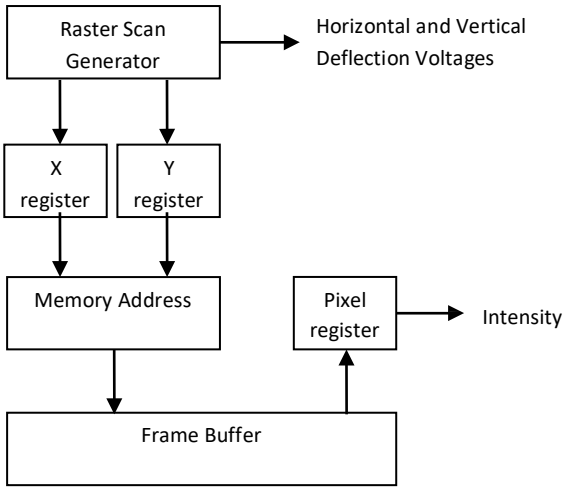


Fig. 1.16: - Basic video controller refresh operation.

- Two registers are used to store the coordinates of the screen pixels which are X and Y
- Initially the X is set to 0 and Y is set to Ymax.
- The value stored in frame buffer for this pixel is retrieved and used to set the intensity of the CRT beam.
- After this X register is incremented by one.
- This procedure is repeated till X becomes equals to Xmax.
- Then X is set to 0 and Y is decremented by one pixel and repeat above procedure.
- This whole procedure is repeated till Y is become equals to 0 and complete the one refresh cycle. Then controller reset the register as top –left corner i.e. X=0 and Y=Ymax and refresh process start for next refresh cycle.
- Since screen must be refreshed at the rate of 60 frames per second the simple procedure illustrated in figure cannot be accommodated by typical RAM chips.
- To speed up pixel processing video controller retrieves multiple values at a time using more numbers of registers and simultaneously refresh block of pixel.
- Such a way it can speed up and accommodate refresh rate more than 60 frames per second.

Raster-graphics system with a display processor

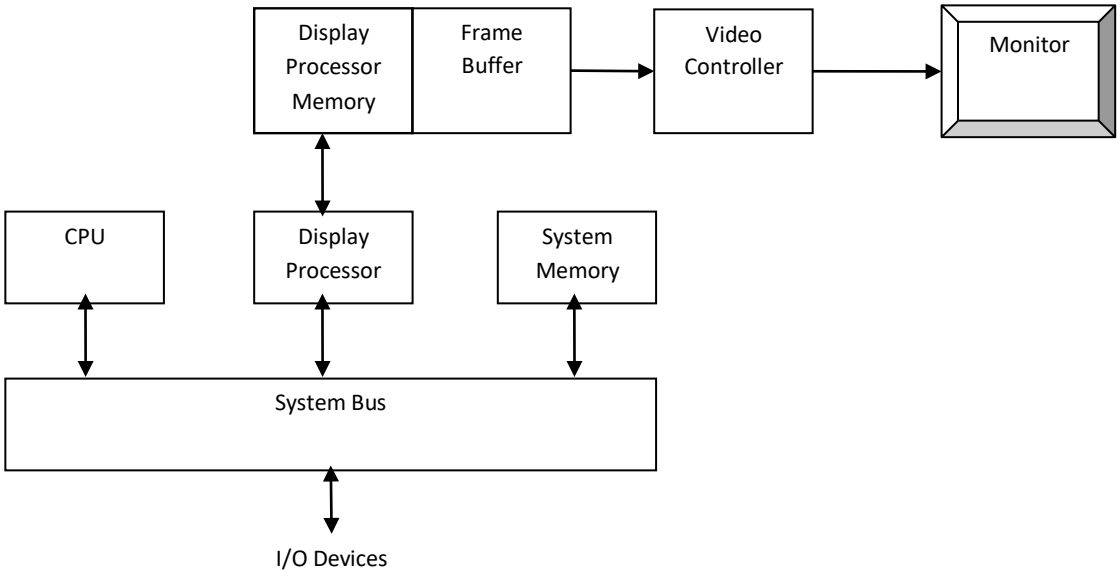


Fig. 1.17: - Architecture of a raster-graphics system with a display processor.

- One way to designing raster system is having separate display coprocessor.
- Purpose of display processor is to free CPU from graphics work.
- Display processors have their own separate memory for fast operation.
- Main work of display processor is digitalizing a picture definition given into a set of pixel intensity values for store in frame buffer.
- This digitalization process is scan conversion.
- Display processor also performs many other functions such as generating various line styles (dashed, dotted, or solid). Display color areas and performing some transformation for manipulating object.
- It also interfaces with interactive input devices such as mouse.
- For reduce memory requirements in raster scan system methods have been devised for organizing the frame buffer as a line list and encoding the intensity information.
- One way to do this is to store each scan line as a set of integer pair one number indicate number of adjacent pixels on the scan line that are having same intensity and second stores intensity value this technique is called run-length encoding.
- A similar approach is when pixel. Intensity is changes linearly, encoded the raster as a set of rectangular areas (cell encoding).
- Disadvantages of encoding is when run length is small it requires more memory then original frame buffer.
- It also difficult for display controller to process the raster when many sort runs are involved.

Random- scan system

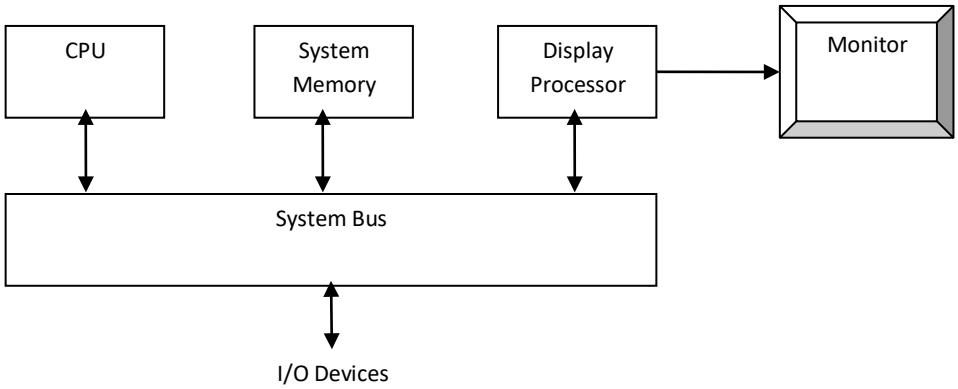


Fig. 1.18: - Architecture of a simple random-scan system.

- An application program is input & stored in the system memory along with a graphics package.
- Graphics commands in the application program are translated by the graphics package into a display file stored in the system memory.
- This display file is used by display processor to refresh the screen.
- Display process goes through each command in display file. Once during every refresh cycle.
- Sometimes the display processor in random scan system is also known as display processing unit or a graphics controller.
- In this system graphics platform are drawn on random scan system by directing the electron beam along the component times of the picture.
- Lines are defined by coordinate end points.
- This input coordinate values are converts to X and Y deflection voltages.
- A scene is then drawn one line at a time.

Graphics input devices

Keyboards

- Keyboards are used as entering text strings. It is efficient devices for inputting such a non-graphics data as picture label.
- Cursor control key's & function keys are common features on general purpose keyboards.
- Many other application of key board which we are using daily used of computer graphics are commanding & controlling through keyboard etc.

Mouse

- Mouse is small size hand-held box used to position screen cursor.
- Wheel or roller or optical sensor is directing pointer on the according to movement of mouse.
- Three buttons are placed on the top of the mouse for signaling the execution of some operation.
- Now a day's more advance mouse is available which are very useful in graphics application for example Z mouse.

Trackball and Spaceball

- Trackball is ball that can be rotated with the finger or palm of the hand to produce cursor movement.

- Potentiometer attached to the ball, measure the amount and direction of rotation.
- They are often mounted on keyboard or Z mouse.
- Space ball provide six-degree of freedom i.e. three dimensional.
- In space ball strain gauges measure the amount of pressure applied to the space ball to provide input for spatial positioning and orientation as the ball is pushed or pulled in various directions.
- Space balls are used in 3D positioning and selection operations in virtual reality system, modeling, animation, CAD and other application.

Joysticks

- A joy stick consists of small vertical lever mounted on a base that is used to steer the screen cursor around.
- Most joy sticks selects screen positioning according to actual movement of stick (lever).
- Some joy sticks are works on pressure applied on sticks.
- Sometimes joy stick mounted on keyboard or sometimes used alone.
- Movement of the stick defines the movement of the cursor.
- In pressure sensitive stick pressure applied on stick decides movement of the cursor. This pressure is measured using strain gauge.
- This pressure sensitive joy sticks also called as isometric joy sticks and they are non movable sticks.

Data glove

- Data glove is used to grasp virtual objects.
- The glow is constructed with series of sensors that detect hand and figure motions.
- Electromagnetic coupling is used between transmitter and receiver antennas which used to provide position and orientation of the hand.
- Transmitter & receiver Antenna can be structured as a set of three mutually perpendicular coils forming 3D Cartesian coordinates system.
- Input from the glove can be used to position or manipulate object in a virtual scene.

Digitizer

- Digitizer is common device for drawing painting or interactively selecting coordinates position on an object.
- One type of digitizers is graphics tablet which input two dimensional coordinates by activating hand cursor or stylus at selected position on a flat surface.
- Stylus is flat pencil shaped device that is pointed at the position on the tablet.

Image Scanner

- Image Scanner scan drawing, graph, color, & black and white photos or text and can stored for computer processing by passing an optical scanning mechanism over the information to be stored.
- Once we have internal representation of a picture we can apply transformation.
- We can also apply various image processing methods to modify the picture.
- For scanned text we can apply modification operation.

Touch Panels

- As name suggest Touch Panels allow displaying objects or screen-position to be selected with the touch or finger.
- A typical application is selecting processing option shown in graphical icons.

- Some system such as a plasma panel are designed with touch screen
- Other system can be adapted for touch input by fitting transparent touch sensing mechanism over a screen.
- Touch input can be recorded with following methods.
  1. Optical methods
  2. Electrical methods
  3. Acoustical methods

Optical method

- Optical touch panel employ a line of infrared LEDs along one vertical and one horizontal edge.
- The opposite edges of the edges containing LEDs are contain light detectors.
- When we touch at a particular position the line of light path breaks and according to that breaking line coordinate values are measured.
- In case two line cuts it will take average of both pixel positions.
- LEDs operate at infrared frequency so it cannot be visible to user.

Electrical method

- An electrical touch panel is constructed with two transparent plates separated by small distance.
- One is coated with conducting material and other is coated with resistive material.
- When outer plate is touch it will come into contact with internal plate.
- When both plates touch it creates voltage drop across the resistive plate that is converted into coordinate values of the selected position.

Acoustical method

- In acoustical touch panel high frequency sound waves are generated in horizontal and vertical direction across a glass plates.
- When we touch the screen the waves from that line are reflected from finger.
- These reflected waves reach again at transmitter position and time difference between sending and receiving is measure and converted into coordinate values.

Light pens

- Light pens are pencil-shaped device used to select positions by detecting light coming from points on the CRT screen.
- Activated light pens pointed at a spot on the screen as the electron beam lights up that spot and generate electronic pulse that causes the coordinate position of the electron beam to be recorded.

Voice systems

- It is used to accept voice command in some graphics workstations.
- It is used to initiate graphics operations.
- It will match input against predefined directory of words and phrases.
- Dictionary is setup for a particular operator by recording his voice.
- Each word is speak several times and then analyze the word and establishes a frequency pattern for that word along with corresponding function need to be performed.
- When operator speaks command it will match with predefine dictionary and perform desired action.

Graphics software and standard

- There are mainly two types of graphics software:
  1. General programming package
  2. Special-purpose application package

General programming package

- A general programming package provides an extensive set of graphics function that can be used in high level programming language such as C or FORTRAN.
- It includes basic drawing element shape like line, curves, polygon, color of element transformation etc.
- Example: - GL (Graphics Library).

Special-purpose application package

- Special-purpose application package are customize for particular application which implement required facility and provides interface so that user need not to vory about how it will work (programming). User can simply use it by interfacing with application.
- Example: - CAD, medical and business systems.

Coordinate representations

- Except few all other general packages are designed to be used with Cartesian coordinate specifications.
- If coordinate values for a picture are specified is some other reference frame they must be converted to Cartesian coordinate before giving input to graphics package.
- Special-purpose package may allow use of other coordinates which suits application.
- In general several different Cartesian reference frames are used to construct and display scene.
- We can construct shape of object with separate coordinate system called modeling coordinates or sometimes local coordinates or master coordinates.
- Once individual object shapes have been specified we can place the objects into appropriate positions called world coordinates.
- Finally the World-coordinates description of the scene is transferred to one or more output device reference frame for display. These display coordinates system are referred to as "**Device Coordinates"** or "**Screen Coordinates".**
- Generally a graphic system first converts the world-coordinates position to normalized device coordinates. In the range from 0 to 1 before final conversion to specific device coordinates.
- An initial modeling coordinates position (Xmc,Ymc) in this illustration is transferred to a device coordinates position(Xdc,Ydc) with the sequence (Xmc,Ymc)→ (Xwc,Ywc)→ (Xnc,Ync)→ (Xdc,Ydc).

Graphic Function

- A general purpose graphics package provides user with Varity of function for creating and manipulating pictures.
- The basic building blocks for pictures are referred to as output primitives. They includes character, string, and geometry entities such as point, straight lines, curved lines, filled areas and shapes defined with arrays of color points.
- Input functions are used for control & process the various input device such as mouse, tablet, etc.
- Control operations are used to controlling and housekeeping tasks such as clearing display screen etc.
- All such inbuilt function which we can use for our purpose are known as graphics function

Software Standard

- Primary goal of standardize graphics software is portability so that it can be used in any hardware systems & avoid rewriting of software program for different system
- Some of these standards are discuss below

Graphical Kernel System (GKS)

- This system was adopted as a first graphics software standard by the international standard organization (ISO) and various national standard organizations including ANSI.
- GKS was originally designed as the two dimensional graphics package and then later extension was developed for three dimensions.

PHIGS (Programmer's Hierarchical Interactive Graphic Standard)

- PHIGS is extension of GKS. Increased capability for object modeling, color specifications, surface rendering, and picture manipulation are provided in PHIGS.
- Extension of PHIGS called **"PHIGS+"** was developed to provide three dimensional surface shading capabilities not available in PHIGS.

Points and Lines

- Point plotting is done by converting a single coordinate position furnished by an application program into appropriate operations for the output device in use.
- Line drawing is done by calculating intermediate positions along the line path between two specified endpoint positions.
- The output device is then directed to fill in those positions between the end points with some color.
- For some device such as a pen plotter or random scan display, a straight line can be drawn smoothly from one end point to other.
- Digital devices display a straight line segment by plotting discrete points between the two endpoints.
- Discrete coordinate positions along the line path are calculated from the equation of the line.
- For a raster video display, the line intensity is loaded in frame buffer at the corresponding pixel positions.
- Reading from the frame buffer, the video controller then plots the screen pixels.
- Screen locations are referenced with integer values, so plotted positions may only approximate actual line positions between two specified endpoints.
- For example line position of $(12.36, 23.87)$ would be converted to pixel position $(12, 24)$.
- This rounding of coordinate values to integers causes lines to be displayed with a stair step appearance ("the jaggies"), as represented in fig 2.1.

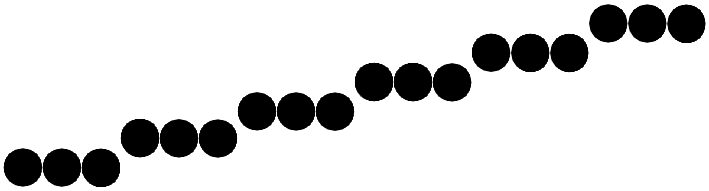


Fig. 2.1: - Stair step effect produced when line is generated as a series of pixel positions.

- The stair step shape is noticeable in low resolution system, and we can improve their appearance somewhat by displaying them on high resolution system.
- More effective techniques for smoothing raster lines are based on adjusting pixel intensities along the line paths.
- For raster graphics device-level algorithms discuss here, object positions are specified directly in integer device coordinates.
- Pixel position will referenced according to scan-line number and column number which is illustrated by following figure.

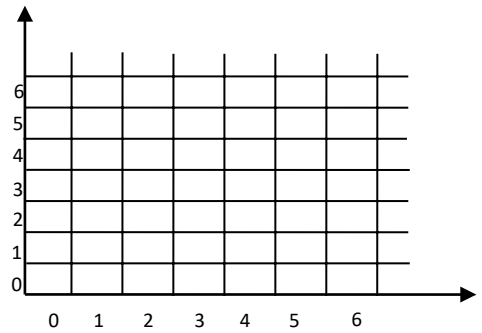


Fig. 2.2: - Pixel positions referenced by scan-line number and column number.

- To load the specified color into the frame buffer at a particular position, we will assume we have available low-level procedure of the form $setpixel(x, y)$.

- Similarly for retrieve the current frame buffer intensity we assume to have procedure $getpixel(x,y)$.

Line Drawing Algorithms

- The Cartesian slop-intercept equation for a straight line is "$y = mx + b$" with 'm' representing slop and 'b' as the intercept.
- The two endpoints of the line are given which are say $(x1, y1)$ and $(x2,y2)$.

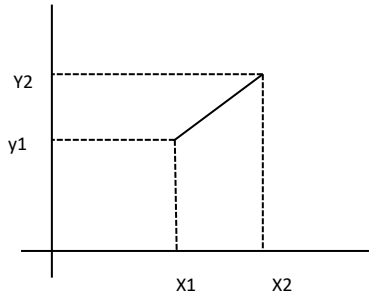


Fig. 2.3: - Line path between endpoint positions.

- We can determine values for the slope m by equation:
  $m = (y2 - y1)/(x2 - x1)$
- We can determine values for the intercept b by equation:
  $b = y1 - m * x1$
- For the given interval Δx along a line, we can compute the corresponding y interval Δy as:
  $\Delta y = m * \Delta x$
- Similarly for Δx:
  $\Delta x = \Delta y/m$
- For line with slop $|m| < 1$, Δx can be set proportional to small horizontal deflection voltage and the corresponding vertical deflection voltage is then set proportional to Δy which is calculated from above equation.
- For line with slop $|m| > 1$, Δy can be set proportional to small vertical deflection voltage and the corresponding horizontal deflection voltage is then set proportional to Δx which is calculated from above equation.
- For line with slop $m = 1$, $\Delta x = \Delta y$ and the horizontal and vertical deflection voltages are equal.

DDA Algorithm

- Digital differential analyzer (DDA) is scan conversion line drawing algorithm based on calculating either Δy or Δx using above equation.
- We sample the line at unit intervals in one coordinate and find corresponding integer values nearest the line path for the other coordinate.
- Consider first a line with positive slope and slope is less than or equal to 1:
  We sample at unit x interval $(\Delta x = 1)$ and calculate each successive y value as follow:
  $y = m * x + b$
  $y_k = m * (x + 1) + b$
  In general $y_k = m * (x + k) + b$, &
  $y_{k+1} = m * (x + k + 1) + b$
  Now write this equation in form:
  $y_{k+1} - y_k = (m * (x + k + 1) + b) - (m * (x + k) + b)$
  $y_{k+1} = y_k + m$
  So that it is computed fast in computer as addition is fast compare to multiplication.

- In above equation k takes integer values starting from 1 and increase by 1 until the final endpoint is reached.
- As m can be any real number between 0 and 1, the calculated y values must be rounded to the nearest integer.
- Consider a case for a line with a positive slope greater than 1:
  We change the role of x and y that is sample at unit y intervals $(\Delta y = 1)$ and calculate each succeeding x value as:
  $x = (y - b)/m$
  $x_1 = ((y + 1) - b)/m$
  In general $x_k = ((y + k) - b)/m$, &
  $x_{k+1} = ((y + k + 1) - b)/m$
  Now write this equation in form:
  $x_{k+1} - x_k = (((y + k + 1) - b)/m) - (((y + k) - b)/m)$
  $x_{k+1} = x_k + 1/m$
- Above both equations are based on the assumption that lines are to be processed from left endpoint to the right endpoint.
- If we processed line from right endpoint to left endpoint than:
  If $\Delta x = -1$ equation become:
  $y_{k+1} = y_k - m$
  If $\Delta y = -1$ equation become:
  $x_{k+1} = x_k - 1/m$
- Above calculated equations also used to calculate pixel position along a line with negative slope.
- **Procedure for DDA line algorithm.**

```
Void lineDDA (int xa, int ya, int xb, int yb)
{
    int dx = xb – xa, dy = yb – ya, steps, k;
    float xincrement, yincrement, x = xa, y = ya;
    if (abs(dx)>abs(dy))
    {
            Steps = abs (dx);
    }
    else
    {
            Steps = abs (dy);
    }
    xincrement = dx/(float) steps;
    yincrement = dy/(float) steps;

    setpixel (ROUND (x), ROUND (y));
    for(k=0;k<steps;k++)
    {
            x += xincrement;
            y += yincrement;
            setpixel (ROUND (x), ROUND (y));
    }
}
```

Advantages of DDA algorithm

- It is faster algorithm.
- It is simple algorithm.

Disadvantage of DDA algorithm

- Floating point arithmetic is time consuming.
- Poor end point accuracy.

Bresenham's Line Algorithm

- An accurate and efficient raster line-generating algorithm, developed by Bresenham which scan converts line using only incremental integer calculations that can be modified to display circles and other curves.
- Figure shows section of display screen where straight line segments are to be drawn.

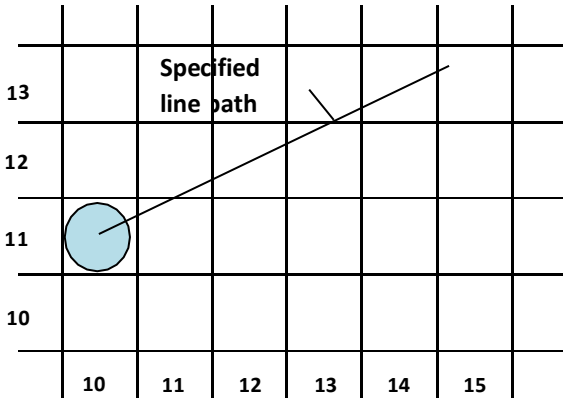


Fig. 2.4: - Section of a display screen where a straight line segment is to be plotted, starting from the pixel at column 10 on scan line 11.

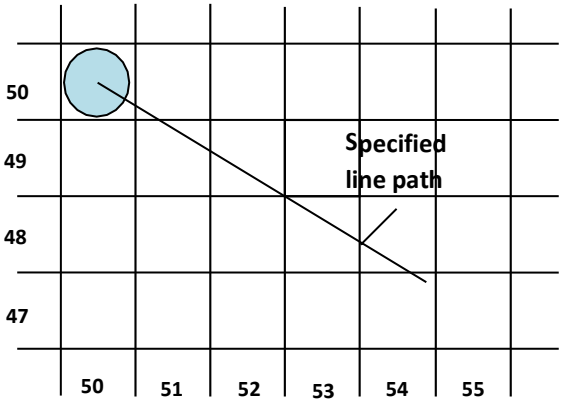


Fig. 2.5: - Section of a display screen where a negative slope line segment is to be plotted, starting from the pixel at column 50 on scan line 50.

- The vertical axes show scan-line positions and the horizontal axes identify pixel column.
- Sampling at unit x intervals in these examples, we need to decide which of two possible pixel position is closer to the line path at each sample step.
- To illustrate bresenham's approach, we first consider the scan-conversion process for lines with positive slope less than 1.
- Pixel positions along a line path are then determined by sampling at unit x intervals.
- Starting from left endpoint (x_0, y_0) of a given line, we step to each successive column and plot the pixel whose scan-line y values is closest to the line path.
- Assuming we have determined that the pixel at (x_k, y_k) is to be displayed, we next need to decide which pixel to plot in column $x_k + 1$.
- Our choices are the pixels at positions $(x_k + 1, y_k)$ and $(x_k + 1, y_k + 1)$.
- Let's see mathematical calculation used to decide which pixel position is light up.
- We know that equation of line is:
  $y = mx + b$
  Now for position $x_k + 1$.
  $y = m(x_k + 1) + b$
- Now calculate distance bet actual line's y value and lower pixel as d_1 and distance bet actual line's y value and upper pixel as d_2.
  $d_1 = y - y_k$

$d_1 = m(x_k + 1) + b - y_k$ ..... (1)

$d_2 = (y_k + 1) - y$

$d_2 = (y_k + 1) - m(x_k + 1) - b$ ..... (2)

- Now calculate $d_1 - d_2$ from equation (1) and (2).

$d_1 - d_2 = (y - y_k) - ((y_k + 1) - y)$

$d_1 - d_2 = \{m(x_k + 1) + b - y_k\} - \{(y_k + 1) - m(x_k + 1) - b\}$

$d_1 - d_2 = \{mx_k + m + b - y_k\} - \{y_k + 1 - mx_k - m - b\}$

$d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1$ ..... (3)

- Now substitute $m = \Delta y/\Delta x$ in equation (3)

$d_1 - d_2 = 2\left(\frac{\Delta y}{\Delta x}\right)(x_k + 1) - 2y_k + 2b - 1$ ..... (4)

- Now we have decision parameter p_k for k^{th} step in the line algorithm is given by:

$p_k = \Delta x(d_1 - d_2)$

$p_k = \Delta x(2\Delta y/\Delta x(x_k + 1) - 2y_k + 2b - 1)$

$p_k = 2\Delta y x_k + 2\Delta y - 2\Delta x y_k + 2\Delta x b - \Delta x$

$p_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + 2\Delta x b - \Delta x$ ..... (5)

$p_k = 2\Delta y x_k - 2\Delta x y_k + C\ (Where\ Constant\ C = 2\Delta y + 2\Delta x b - \Delta x)$ ..... (6)

- The sign of p_k is the same as the sign of $d_1 - d_2$, since $\Delta x > 0$ for our example.
- Parameter c is constant which is independent of pixel position and will eliminate in the recursive calculation for p_k.
- Now if p_k is negative then we plot the lower pixel otherwise we plot the upper pixel.
- So successive decision parameters using incremental integer calculation as:

$p_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + C$

- Now Subtract pk from p_{k+1}

$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$

$p_{k+1} - p_k = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + C - 2\Delta y x_k + 2\Delta x y_k - C$

But $x_{k+1} = x_k + 1$, so that $(x_{k+1} - x_k) = 1$

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

- Where the terms $y_{k+1} - y_k$ is either 0 or 1, depends on the sign of parameter p_k.
- This recursive calculation of decision parameters is performed at each integer x position starting at the left coordinate endpoint of the line.
- The first decision parameter p_0 is calculated using equation (5) as first time we need to take constant part into account so:

$p_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + 2\Delta x b - \Delta x$

$p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x b - \Delta x$

Now $Substitute\ b = y_0 - mx_0$

$p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x(y_0 - mx_0) - \Delta x$

$Now\ Substitute\ m = \Delta y/\Delta x$

$p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x(y_0 - (\Delta y/\Delta x)x_0) - \Delta x$

$p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x y_0 - 2\Delta y x_0 - \Delta x$

$p_0 = 2\Delta y - \Delta x$

- Let's see Bresenham's line drawing algorithm for $|m| < 1$
  1. Input the two line endpoints and store the left endpoint in (x_0, y_0).
  2. Load (x_0, y_0) into the frame buffer; that is, plot the first point.
  3. Calculate constants Δx, Δy, $2\Delta y$, and $2\Delta y - 2\Delta x$, 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 test:
   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.

- Bresenham's algorithm is generalized to lines with arbitrary slope by considering symmetry between the various octants and quadrants of the xy plane.
- For lines with positive slope greater than 1 we interchange the roles of the x and y directions.
- Also we can revise algorithm to draw line from right endpoint to left endpoint, both x and y decrease as we step from right to left.
- When $d_1 - d_2 = 0$ we choose either lower or upper pixel but once we choose lower than for all such case for that line choose lower and if we choose upper the for all such case choose upper.
- For the negative slope the procedure are similar except that now one coordinate decreases as the other increases.
- The special case handle separately. Horizontal line $(\Delta y = 0)$, vertical line $(\Delta x = 0)$ and diagonal line with $|\Delta x| = |\Delta y|$ each can be loaded directly into the frame buffer without processing them through the line plotting algorithm.

Parallel Execution of Line Algorithms

- The line-generating algorithms we have discussed so far determine pixel positions sequentially.
- With parallel computer we can calculate pixel position along a line path simultaneously by dividing work among the various processors available.
- One way to use multiple processors is partitioning existing sequential algorithm into small parts and compute separately.
- Alternatively we can go for other ways to setup the processing so that pixel positions can be calculated efficiently in parallel.
- Important point to be taking into account while devising parallel algorithm is to balance the load among the available processors.
- Given $\boldsymbol{n_p}$ number of processors we can set up parallel Bresenham line algorithm by subdividing the line path into $\boldsymbol{n_p}$ partitions and simultaneously generating line segment in each of the subintervals.
- For a line with slope $0 < m < 1$ and left endpoint coordinate position (x_0, y_0), we partition the line along the positive x direction.
- The distance between beginning x positions of adjacent partitions can be calculated as:
  $\Delta x_p = (\Delta x + n_p - 1)/n_p$
  Were Δx is the width of the line. And value for partition with Δx_p is computed using integer division.
- Numbering the partitions and the processors, as $0, 1, 2$, up to $\boldsymbol{n_p - 1}$, we calculate the starting x coordinate for the k^{th} partition as:
  $x_k = x_0 + k\Delta x_p$
- To apply Bresenham's algorithm over the partitions, we need the initial value for the y coordinate and the initial value for the decision parameter in each partition.
- The change Δy_p in the y direction over each partition is calculated from the line slope m and partition width Δx_p:
- $\Delta y_p = m\Delta x_p$

- At the k^{th} partition, the starting y coordinate is then
- $y_k = y_0 + round(k\Delta y_p)$
- The initial decision parameter for Bresenham's algorithm at the start of the k^{th} subinterval is obtained from Equation(6):

  $p_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + 2\Delta x b - \Delta x$

  $p_k = 2\Delta y(x_0 + k\Delta x_p) - 2\Delta x(y_0 + round(k\Delta y_p)) + 2\Delta y + 2\Delta x(y_0 - \frac{\Delta y}{\Delta x}x_0) - \Delta x$

  $p_k = 2\Delta y x_0 - 2\Delta y k\Delta x_p - 2\Delta x y_0 - 2\Delta x round(k\Delta y_p) + 2\Delta y + 2\Delta x y_0 - 2\Delta y x_0 - \Delta x$

  $p_k = 2\Delta y k\Delta x_p - 2\Delta x round(k\Delta y_p) + 2\Delta y - \Delta x$
- Each processor then calculates pixel positions over its assigned subinterval.
- The extension of the parallel Bresenham algorithm to a line with slope greater than 1 is achieved by partitioning the line in the y direction and calculating beginning x values for the positions.
- For negative slopes, we increment coordinate values in one direction and decrement in the other.

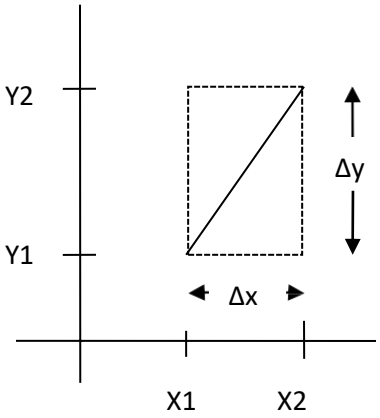


Fig. 2.6: - Bounding box for a line with coordinate extents Δx and Δy.

- Another way to set up parallel algorithms on raster system is to assign each processor to a particular group of screen pixels.
- With sufficient number of processor we can assign each processor to one pixel within some screen region.
- This approach can be adapted to line display by assigning one processor to each of the pixels within the limit of the bounding rectangle and calculating pixel distance from the line path.
- The number of pixels within the bounding rectangle of a line is $\Delta x \times \Delta y$.
- Perpendicular distance d from line to a particular pixel is calculated by:

  $d = Ax + By + C$

  Where

  $A = -\Delta y/linelength$

  $B = -\Delta x/linelength$

  $C = (x_0\Delta y - y_0\Delta x)/linelength$

  With

  $linelength = \sqrt{\Delta x^2 + \Delta y^2}$
- Once the constant A, B, and C have been evaluated for the line each processors need to perform two multiplications and two additions to compute the pixel distance d.
- A pixel is plotted if d is less than a specified line thickness parameter.
- Instead of partitioning the screen into single pixels, we can assign to each processor either a scan line or a column a column of pixels depending on the line slope.

- Each processor calculates line intersection with horizontal row or vertical column of pixels assigned to that processor.
- If vertical column is assign to processor then x is fix and it will calculate y and similarly is horizontal row is assign to processor then y is fix and x will be calculated.
- Such direct methods are slow in sequential machine but we can perform very efficiently using multiple processors.

Circle

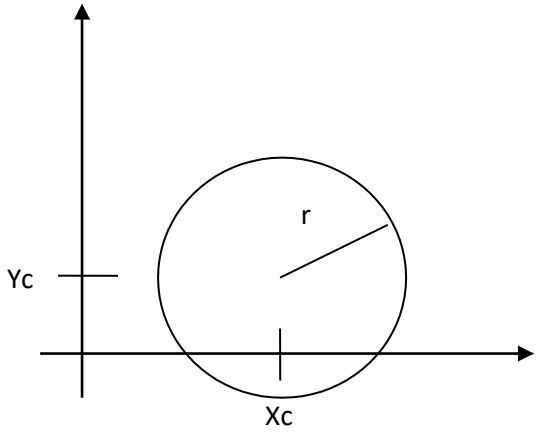


Fig. 2.7: - Circle with center coordinates (x_c, y_c) and radius r.

- A circle is defined as the set of points that are all at a given distance r from a center position say (x_c, y_c).

Properties of Circle

- The distance relationship is expressed by the Pythagorean theorem in Cartesian coordinates as:
  $(x - x_c)^2 + (y - y_c)^2 = r^2$
- We could use this equation to calculate circular boundary points by incrementing 1 in x direction in every steps from $x_c - r$ to $x_c + r$ and calculate corresponding y values at each position as:
  $(x - x_c)^2 + (y - y_c)^2 = r^2$
  $(y - y_c)^2 = r^2 - (x - x_c)^2$
  $(y - y_c) = \pm\sqrt{r^2 - (x_c - x)^2}$
  $y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$
- But this is not best method for generating a circle because it requires more number of calculations which take more time to execute.
- And also spacing between the plotted pixel positions is not uniform as shown in figure below.

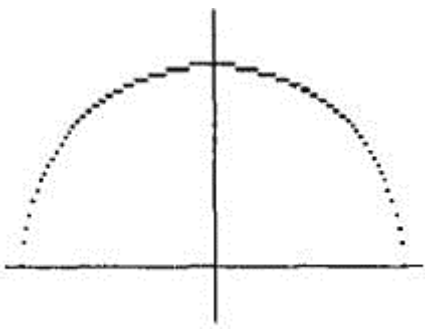


Fig. 2.8: - Positive half of circle showing non uniform spacing bet calculated pixel positions.

- We can adjust spacing by stepping through y values and calculating x values whenever the absolute value of the slop of the circle is greater than 1. But it will increases computation processing requirement.
- Another way to eliminate the non-uniform spacing is to draw circle using polar coordinates 'r' and 'θ'.
- Calculating circle boundary using polar equation is given by pair of equations which is as follows.
  $x = x_c + r\cos\theta$
  $y = y_c + r\sin\theta$
- When display is produce using these equations using fixed angular step size circle is plotted with uniform spacing.
- The step size 'θ' is chosen according to application and display device.
- For a more continuous boundary on a raster display we can set the step size at $1/r$. This plot pixel position that are approximately one unit apart.
- Computation can be reduced by considering symmetry city property of circles. The shape of circle is similar in each quadrant.
- We can obtain pixel position in second quadrant from first quadrant using reflection about y axis and similarly for third and fourth quadrant from second and first respectively using reflection about x axis.
- We can take one step further and note that there is also symmetry between octants. Circle sections in adjacent octant within one quadrant are symmetric with respect to the 45^0 line dividing the two octants.
- This symmetry condition is shown in figure below where point (x, y) on one circle sector is mapped in other seven sector of circle.

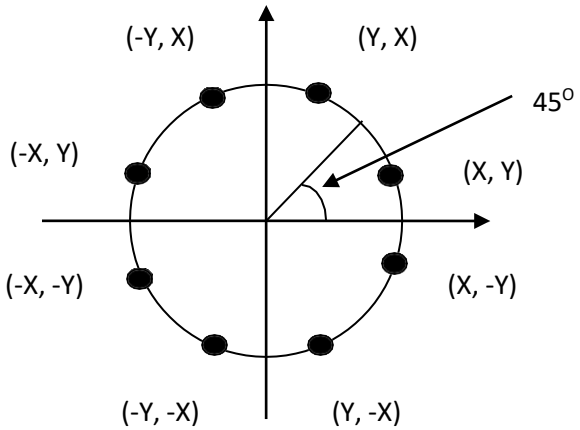


Fig. 2.9: - symmetry of circle.

- Taking advantage of this symmetry property of circle we can generate all pixel position on boundary of circle by calculating only one sector from $x = 0$ to $x = y$.
- Determining pixel position along circumference of circle using any of two equations shown above still required large computation.
- More efficient circle algorithm are based on incremental calculation of decision parameters, as in the Bresenham line algorithm.
- Bresenham's line algorithm can be adapted to circle generation by setting decision parameter for finding closest pixel to the circumference at each sampling step.
- The Cartesian coordinate circle equation is nonlinear so that square root evaluations would be required to compute pixel distance from circular path.
- Bresenham's circle algorithm avoids these square root calculation by comparing the square of the pixel separation distance.

- A method for direct distance comparison to test the midpoint between two pixels to determine if this midpoint is inside or outside the circle boundary.
- This method is easily applied to other conics also.
- Midpoint approach generates same pixel position as generated by bresenham's circle algorithm.
- The error involve in locating pixel positions along any conic section using midpoint test is limited to one-half the pixel separation.

Midpoint Circle Algorithm

- Similar to raster line algorithm we sample at unit interval and determine the closest pixel position to the specified circle path at each step.
- Given radius 'r' and center (x_c, y_c)
- We first setup our algorithm to calculate circular path coordinates for center (0, 0). And then we will transfer calculated pixel position to center (x_c, y_c) by adding x_c to x and y_c to y.
- Along the circle section from $x = 0$ to $x = y$ in the first quadrant, the slope of the curve varies from 0 to -1 so we can step unit step in positive x direction over this octant and use a decision parameter to determine which of the two possible y position is closer to the circular path.
- Position in the other seven octants are then obtain by symmetry.
- For the decision parameter we use the circle function which is:
  $f_{circle}(x, y) = x^2 + y^2 - r^2$
- Any point which is on the boundary is satisfied $f_{circle}(x, y) = 0$ if the point is inside circle function value is negative and if point is outside circle the function value is positive which can be summarize as below.

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

- Above equation we calculate for the mid positions between pixels near the circular path at each sampling step and we setup incremental calculation for this function as we did in the line algorithm.
- Below figure shows the midpoint between the two candidate pixels at sampling position $x_k + 1$.

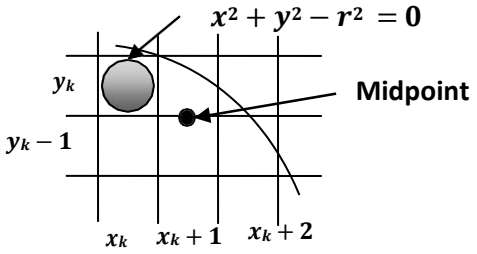


Fig. 2.10: - Midpoint between candidate pixel at sampling position $x_k + 1$ along circle path.

- Assuming we have just plotted the pixel at (x_k, y_k) and next we need to determine whether the pixel at position '$(x_k + 1, y_k)$' or the one at position' $(x_k + 1, y_k - 1)$' is closer to circle boundary.
- So for finding which pixel is more closer using decision parameter evaluated at the midpoint between two candidate pixels as below:

$$p_k = f_{circle}\left(x_k + 1, y_k - \frac{1}{2}\right)$$

$$p_k = (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

- If $p_k < 0$ this midpoint is inside the circle and the pixel on the scan line y_k is closer to circle boundary. Otherwise the midpoint is outside or on the boundary and we select the scan line $y_k - 1$.
- Successive decision parameters are obtain using incremental calculations as follows:

$$p_{k+1} = f_{circle}\left(x_{k+1}+1, y_{k+1}-\frac{1}{2}\right)$$

$$p_{k+1} = [(x_k+1)+1]^2 + \left(y_{k+1}-\frac{1}{2}\right)^2 - r^2$$

- Now we can obtain recursive calculation using equation of p_{k+1} and p_k as follow.

$$p_{k+1} - p_k = \left([(x_k+1)+1]^2 + \left(y_{k+1}-\frac{1}{2}\right)^2 - r^2\right) - \left((x_k+1)^2 + \left(y_k-\frac{1}{2}\right)^2 - r^2\right)$$

$$p_{k+1} - p_k = (x_k+1)^2 + 2(x_k+1) + 1 + {y_{k+1}}^2 - y_{k+1} + \frac{1}{4} - r^2 - (x_k+1)^2 - {y_k}^2 + y_k - \frac{1}{4} + r^2$$

$$p_{k+1} - p_k = 2(x_k+1) + 1 + {y_{k+1}}^2 - y_{k+1} - {y_k}^2 + y_k$$

$$p_{k+1} - p_k = 2(x_k+1) + ({y_{k+1}}^2 - {y_k}^2) - (y_{k+1} - y_k) + 1$$

$$p_{k+1} = p_k + 2(x_k+1) + ({y_{k+1}}^2 - {y_k}^2) - (y_{k+1} - y_k) + 1$$

- In above equation y_{k+1} is either y_k or $y_k - 1$ depending on the sign of the p_k.
- Now we can put $2x_{k+1} = 2x_k + 2$ and when we select $y_{k+1} = y_k - 1$ we can obtain $2y_{k+1} = 2y_k - 2$.
- The initial decision parameter is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$ as follows.

$$p_0 = f_{circle}\left(0+1, r-\frac{1}{2}\right)$$

$$p_0 = 1^2 + \left(r-\frac{1}{2}\right)^2 - r^2$$

$$p_0 = 1 + r^2 - r + \frac{1}{4} - r^2$$

$$p_0 = \frac{5}{4} - r$$

Algorithm for Midpoint Circle Generation

1. Input radius r and circle center (x_c, y_c), and obtain the first point on the circumference of a circle centered on the origin as
$$(x_0, y_0) = (0, r)$$
2. calculate the initial value of the decision parameter as
$$p_0 = \frac{5}{4} - r$$
3. At each x_k position, starting at $k = 0$, perform the following test:
If $p_k < 0$, the next point along the circle centered on $(0, 0)$ is $(x_k + 1, y_k)$ &
$$p_{k+1} = p_k + 2x_{k+1} + 1$$
Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ &
$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$
Where $2x_{k+1} = 2x_k + 2$, & $2y_{k+1} = 2y_k - 2$.
4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the coordinate values:
$$x = x + x_c, y = y + y_c$$
6. Repeat steps 3 through 5 until $x \geq y$.

Ellipse

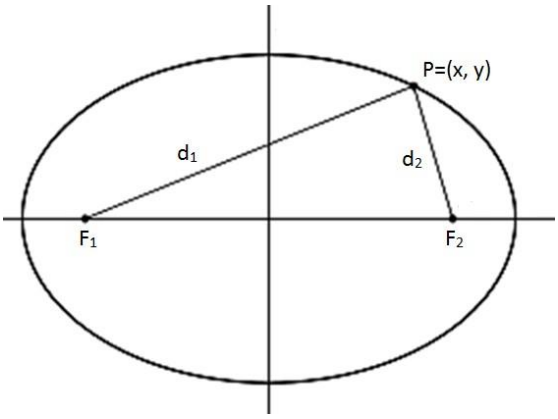


Fig. 2.11: - Ellipse generated about foci f_1 and f_2.

- AN ellipse is defined as the set of points such that the sum of the distances from two fixed positions (**foci**) is same for all points.

Properties of Ellipse

- If we labeled distance from two foci to any point on ellipse boundary as d_1 and d_2 then the general equation of an ellipse can be written as follow.
  $d_1 + d_2 = Constant$
- Expressing distance in terms of focal coordinates $f_1 = (x_1, y_1)$ and $f_2 = (x_2, y_2)$ we have
  $\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = Constant$
- An interactive method for specifying an ellipse in an arbitrary orientation is to input two foci and a point on the ellipse boundary.
- With this three coordinates we can evaluate constant in equation:
- $\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = Constant$
- We can also write this equation in the form
  $Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$
- Where the coefficients $A, B, C, D, E,$ and F are evaluated in terms of the focal coordinates and the dimensions of the major and minor axes of the ellipse.
- Major axis of an ellipse is straight line segment passing through both foci and extends up to boundary on both sides.
- The minor axis spans shortest dimension of ellipse, it bisect the major axis at right angle in two equal half.
- Then coefficient in $Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$ can be evaluated and used to generate pixels along the elliptical path.
- Ellipse equation are greatly simplified if we align major and minor axis with coordinate axes i.e. $x - axis$ and $y - axis$.
- We can say ellipse is in standard position if their major and minor axes are parallel to $x - axis$ and $y - axis$ which is shown in below figure.

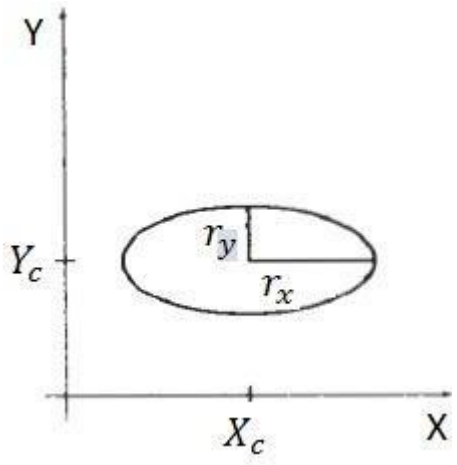


Fig. 2.12: - Ellipse centered at (x_c, y_c) with semi major axis r_x and semi minor axis r_y are parallel to coordinate axis.

- Equation of ellipse shown in figure 2.12 can be written in terms of the ellipse center coordinates and parameters r_x and r_y as.

$$\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{y - y_c}{r_y}\right)^2 = 1$$

- Using the polar coordinates r and θ, we can also describe the ellipse in standard position with the parametric equations:

$$x = x_c + r_x \cos\theta$$
$$y = y_c + r_y \sin\theta$$

- Symmetry considerations can be used to further reduced computations.
- An ellipse in standard position is symmetric between quadrants but unlike a circle it is not symmetric between octant.
- Thus we must calculate boundary point for one quadrant and then other three quadrants point can be obtained by symmetry as shown in figure below.

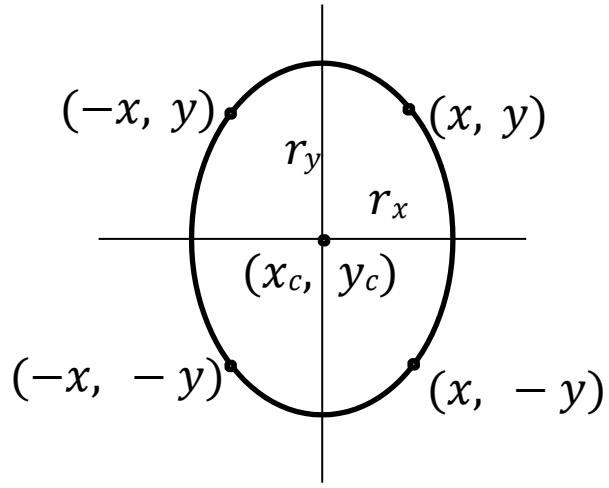


Fig. 2.13: - symmetry of an ellipse.

Midpoint Ellipse Algorithm

- Midpoint ellipse algorithm is a method for drawing ellipses in computer graphics. This method is modified from Bresenham's algorithm.

- The advantage of this modified method is that only addition operations are required in the program loops.
- This leads to simple and fast implementation in all processors.
- Given parameters r_x, r_y and (x_c, y_c) we determine points (x, y) for an ellipse in standard position centered on the origin, and then we shift the points so the ellipse is centered at (x_c, y_c).
- If we want to display the ellipse in nonstandard position then we rotate the ellipse about its center to align with required direction.
- For the present we consider only the standard position.
- In this method we divide first quadrant into two parts according to the slope of an ellipse as shown in figure below.

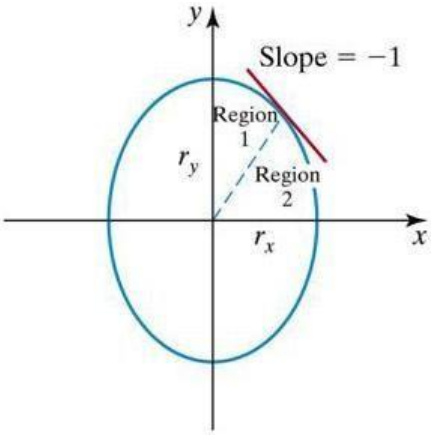


Fig. 2.14: - Ellipse processing regions. Over the region 1 the magnitude of ellipse slope is < 1 and over the region 2 the magnitude of ellipse slope > 1.

- We take unit step in x direction if magnitude of slope is less than 1 in that region otherwise we take unit step in y direction.
- Boundary divides region at $slope = -1$.
- With $r_x < r_y$ we process this quadrant by taking unit steps in x direction in region 1 and unit steps in y direction in region 2.
- Region 1 and 2 can be processed in various ways.
- We can start from $(0, r_y)$ and step clockwise along the elliptical path in the first quadrant shifting from unit step in x to unit step in y when slope becomes less than -1.
- Alternatively, we could start at $(r_x, 0)$ and select points in a counterclockwise order, shifting from unit steps in y to unit steps in x when the slope becomes greater than -1.
- With parallel processors, we could calculate pixel positions in the two regions simultaneously.
- Here we consider sequential implementation of midpoint algorithm. We take the start position at $(0, r_y)$ and steps along the elliptical path in clockwise order through the first quadrant.
- We define ellipse function for center of ellipse at $(0, 0)$ as follows:

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

- Which has the following properties:

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

- Thus the ellipse function serves as the decision parameter in the midpoint ellipse algorithm.
- At each sampling position we select the next pixel from two candidate pixel.

- Starting at $(0, r_y)$ we take unit step in x direction until we reach the boundary between region 1 and 2 then we switch to unit steps in y direction in remaining portion on ellipse in first quadrant.
- At each step we need to test the value of the slope of the curve for deciding the end point of the region-1.
- The ellipse slope is calculated using following equation.
  $$\frac{dy}{dx} = -\frac{2r_y{}^2x}{2r_x{}^2y}$$
- At boundary between region 1 and 2 $slope = -1$ and equation become.
  $$2r_y{}^2x = 2r_x{}^2y$$
- Therefore we move out of region 1 whenever following equation is false
  $$2r_y{}^2x \leq 2r_x{}^2y$$
- Following figure shows the midpoint between the two candidate pixels at sampling position $x_k + 1$ in the first region.

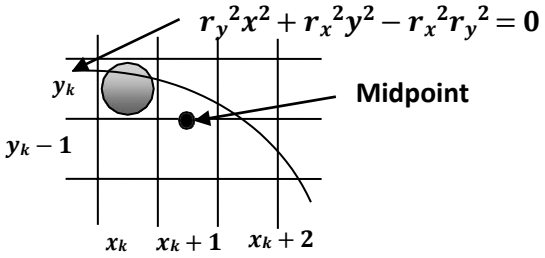


Fig. 2.15: - Midpoint between candidate pixels at sampling position $x_k + 1$ along an elliptical path.

- Assume we are at (x_k , y_k) position and we determine the next position along the ellipse path by evaluating decision parameter at midpoint between two candidate pixels.
  $$p1_k = f_{ellipse}\left(x_k + 1, y_k - \frac{1}{2}\right)$$
  $$p1_k = r_y{}^2(x_k + 1)^2 + r_x{}^2\left(y_k - \frac{1}{2}\right)^2 - r_x{}^2r_y{}^2$$

- If $p1_k < 0$, the midpoint is inside the ellipse and the pixel on scan line y_k is closer to ellipse boundary otherwise the midpoint is outside or on the ellipse boundary and we select the pixel $y_k - 1$.
- At the next sampling position decision parameter for region 1 is evaluated as.
  $$p1_{k+1} = f_{ellipse}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$
  $$p1_{k+1} = r_y{}^2[(x_k + 1) + 1]^2 + r_x{}^2\left(y_{k+1} - \frac{1}{2}\right)^2 - r_x{}^2r_y{}^2$$

- Now subtract $p1_k$ from $p1_{k+1}$

$$p1_{k+1} - p1_k = r_y{}^2[(x_k + 1) + 1]^2 + r_x{}^2\left(y_{k+1} - \frac{1}{2}\right)^2 - r_x{}^2 r_y{}^2 - r_y{}^2(x_k + 1)^2 - r_x{}^2\left(y_k - \frac{1}{2}\right)^2 + r_x{}^2 r_y{}^2$$

$$p1_{k+1} - p1_k = r_y{}^2[(x_k + 1) + 1]^2 + r_x{}^2\left(y_{k+1} - \frac{1}{2}\right)^2 - r_y{}^2(x_k + 1)^2 - r_x{}^2\left(y_k - \frac{1}{2}\right)^2$$

$$p1_{k+1} - p1_k = r_y{}^2(x_k + 1)^2 + 2r_y{}^2(x_k + 1) + r_y{}^2 + r_x{}^2\left(y_{k+1} - \frac{1}{2}\right)^2 - r_y{}^2(x_k + 1)^2 - r_x{}^2\left(y_k - \frac{1}{2}\right)^2$$

$$p1_{k+1} - p1_k = 2r_y{}^2(x_k + 1) + r_y{}^2 + r_x{}^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right]$$

- Now making $p1_{k+1}$ as subject.

$$p1_{k+1} = p1_k + 2r_y{}^2(x_k + 1) + r_y{}^2 + r_x{}^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right]$$

- Here y_{k+1} is either y_k or $y_k - 1$, depends on the sign of $p1_k$
- Now we calculate the initial decision parameter $p1_0$ by putting $(x_0, y_0) = (0, r_y)$ as follow.

$$p1_0 = f_{ellipse}\left(0 + 1, r_y - \frac{1}{2}\right)$$

$$p1_0 = r_y{}^2(1)^2 + r_x{}^2\left(r_y - \frac{1}{2}\right)^2 - r_x{}^2 r_y{}^2$$

$$p1_0 = r_y{}^2 + r_x{}^2\left(r_y - \frac{1}{2}\right)^2 - r_x{}^2 r_y{}^2$$

$$p1_0 = r_y{}^2 - r_x{}^2 r_y + \frac{1}{4}r_x{}^2$$

- Now we similarly calculate over region 2 by unit stepping in negative y direction and the midpoint is now taken between horizontal pixels at each step as shown in figure below.

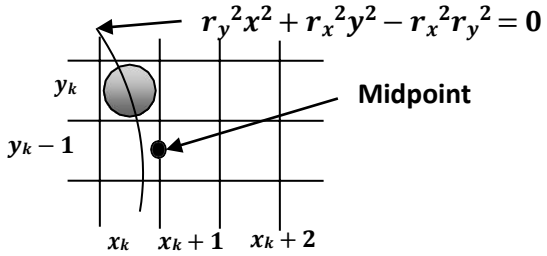


Fig. 2.16: - Midpoint between candidate pixels at sampling position $y_k - 1$ along an elliptical path.

- For this region, the decision parameter is evaluated as follows.

$$p2_k = f_{ellipse}\left(x_k + \frac{1}{2}, y_k - 1\right)$$

$$p2_k = r_y{}^2\left(x_k + \frac{1}{2}\right)^2 + r_x{}^2(y_k - 1)^2 - r_x{}^2 r_y{}^2$$

If $p2_k > 0$ the midpoint is outside the ellipse boundary, and we select the pixel at x_k.

- If $p2_k \leq 0$ the midpoint is inside or on the ellipse boundary and we select $x_k + 1$.
- At the next sampling position decision parameter for region 2 is evaluated as.

$$p2_{k+1} = f_{ellipse}\left(x_{k+1} + \frac{1}{2}, y_{k+1} - 1\right)$$

$$p2_{k+1} = r_y{}^2\left(x_{k+1} + \frac{1}{2}\right)^2 + r_x{}^2[(y_k - 1) - 1]^2 - r_x{}^2 r_y{}^2$$

- Now subtract $p2_k$ from $p2_{k+1}$

$$p2_{k+1} - p2_k = r_y{}^2\left(x_{k+1} + \frac{1}{2}\right)^2 + r_x{}^2[(y_k - 1) - 1]^2 - r_x{}^2 r_y{}^2 - r_y{}^2\left(x_k + \frac{1}{2}\right)^2 - r_x{}^2(y_k - 1)^2 + r_x{}^2 r_y{}^2$$

$$p2_{k+1} - p2_k = r_y{}^2\left(x_{k+1} + \frac{1}{2}\right)^2 + r_x{}^2(y_k - 1)^2 - 2r_x{}^2(y_k - 1) + r_x{}^2 - r_y{}^2\left(x_k + \frac{1}{2}\right)^2 - r_x{}^2(y_k - 1)^2$$

$$p2_{k+1} - p2_k = r_y{}^2\left(x_{k+1} + \frac{1}{2}\right)^2 - 2r_x{}^2(y_k - 1) + r_x{}^2 - r_y{}^2\left(x_k + \frac{1}{2}\right)^2$$

$$p2_{k+1} - p2_k = -2r_x{}^2(y_k - 1) + r_x{}^2 + r_y{}^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right]$$

Here x_{k+1} is either x_k or $x_k + 1$, depends on the sign of $p2_k$.

- In region 2 initial position is selected which is last position of region one and the initial decision parameter is calculated as follows.

$$p2_0 = f_{ellipse}\left(x_0 + \frac{1}{2}, y_0 - 1\right)$$

$$p2_0 = r_y{}^2\left(x_0 + \frac{1}{2}\right)^2 + r_x{}^2(y_0 - 1)^2 - r_x{}^2 r_y{}^2$$

- For simplify calculation of $p2_0$ we could select pixel position in counterclockwise order starting at $(r_x, 0)$.
- In above case we take unit step in the positive y direction up to the last point selected in region 1

Algorithm for Midpoint Ellipse Generation

1. Input r_x, r_y and ellipse center (x_c, y_c), and obtain the first point on an ellipse centered on the origin as
$$(x_0, y_0) = (0, r_y)$$
2. Calculate the initial value of the decision parameter in region 1 as
$$p1_0 = r_y{}^2 - r_x{}^2 r_y{}^2 + \frac{1}{4} r_x{}^2$$
3. At each x_k position in region 1, starting at $k = 0$, perform the following test:

   If $p1_k < 0$, the next point along the ellipse centered on $(0, 0)$ is (x_{k+1}, y_k) and
$$p1_{k+1} = p1_k + 2r_y{}^2 x_{k+1} + r_y{}^2$$

   Otherwise, the next point along the ellipse is $(x_{k+1}, y_k - 1)$ and
$$p1_{k+1} = p1_k + 2r_y{}^2 x_{k+1} + r_y{}^2 - 2r_x{}^2 y_{k+1}$$
   With
$$2r_y{}^2 x_{k+1} = 2r_y{}^2 x_k + 2r_y{}^2$$
$$2r_x{}^2 y_{k+1} = 2r_x{}^2 y_k - 2r_x{}^2$$
   And continue until $2r_y{}^2 x \leq 2r_x{}^2 y$
4. Calculate the initial value of the decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as
$$p2_0 = r_y{}^2 \left(x_0 + \frac{1}{2}\right)^2 + r_x{}^2 (y_0 - 1)^2 - r_x{}^2 r_y{}^2$$
5. At each y_k position in region-2, starting at $k = 0$, perform the following test:

   If $p2_k > 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_k, y_k - 1)$ and
$$p2_{k+1} = p2_k - 2r_x{}^2 y_{k+1} + r_x{}^2$$
   Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and
$$p2_{k+1} = p2_k - 2r_x{}^2 y_{k+1} + r_x{}^2 + 2r_y{}^2 x_{k+1}$$
   Using the same incremental calculations for x and y as in region 1.
6. Determine symmetry points in the other three quadrants.
7. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values:
$$x = x + x_c$$
$$y = y + y_c$$
   Repeat the steps for region 2 until $y_k \geq 0$.

Filled-Area Primitives

- In practical we often use polygon which are filled with some color or pattern inside it.

- There are two basic approaches to area filling on raster systems.
- One way to fill an area is to determine the overlap intervals for scan line that cross the area.
- Another method is to fill the area is to start from a given interior position and paint out wards from this point until we encounter boundary.

Scan-Line Polygon Fill Algorithm

- Figure below shows the procedure for scan-line filling algorithm.

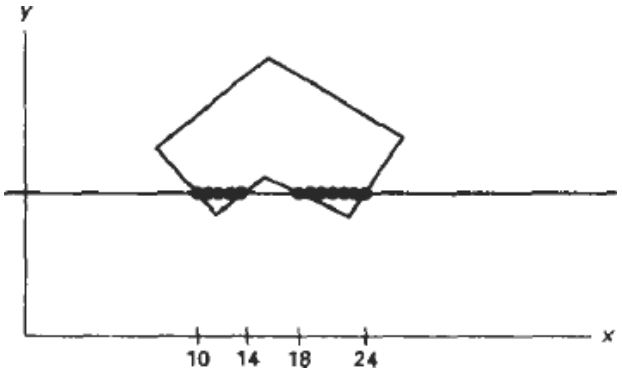


Fig. 2.17: - Interior pixels along a scan line passing through a polygon area.

- For each scan-line crossing a polygon, the algorithm locates the intersection points are of scan line with the polygon edges.
- This intersection points are stored from left to right.
- Frame buffer positions between each pair of intersection point are set to specified fill color.
- Some scan line intersects at vertex position they are required special handling.
- For vertex we must look at the other endpoints of the two line segments of the polygon which meet at this vertex.
- If these points lie on the same (up or down) side of the scan line, then that point is counts as two intersection points.
- If they lie on opposite sides of the scan line, then the point is counted as single intersection.
- This is illustrated in figure below

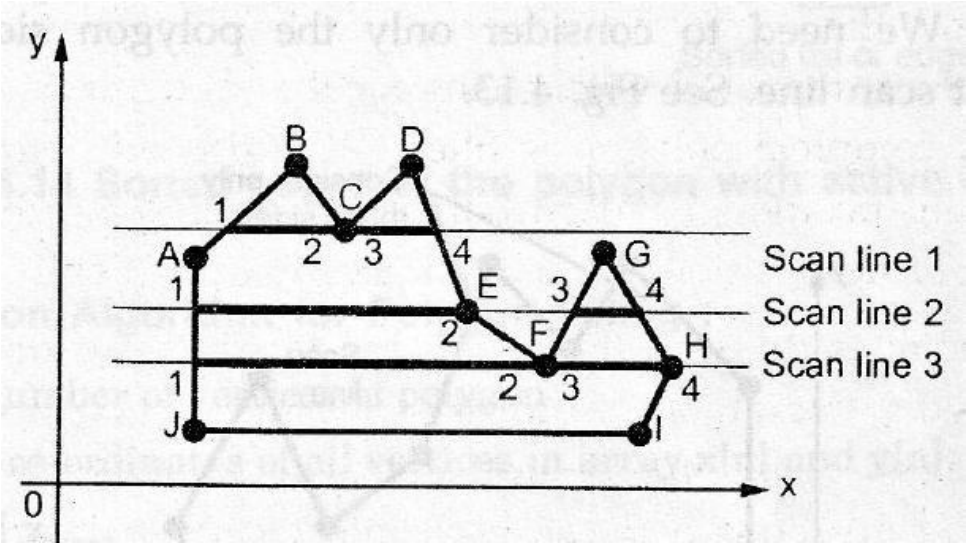


Fig. 2.18: - Intersection points along the scan line that intersect polygon vertices.

- As shown in the Fig. 2.18, each scan line intersects the vertex or vertices of the polygon. For scan line 1, the other end points (B and D) of the two line segments of the polygon lie on the same side of the scan

line, hence there are two intersections resulting two pairs: 1 -2 and 3 - 4. Intersections points 2 and 3 are actually same Points. For scan line 2 the other endpoints (D and F) of the two line segments of the Polygon lie on the opposite sides of the scan line, hence there is a single intersection resulting two pairs: l - 2 and 3 - 4. For scan line 3, two vertices are the intersection points"

- For vertex F the other end points E and G of the two line segments of the polygon lie on the same side of the scan line whereas for vertex H, the other endpoints G and I of the two line segments of the polygon lie on the opposite side of the scan line. Therefore, at vertex F there are two intersections and at vertex H there is only one intersection. This results two pairs: 1 - 2 and 3 - 4 and points 2 and 3 are actually same points.
- Coherence methods often involve incremental calculations applied along a single scan line or between successive scan lines.
- In determining edge intersections, we can set up incremental coordinate calculations along any edge by exploiting the fact that the slope of the edge is constant from one scan line to the next.
- Figure below shows three successive scan-lines crossing the left edge of polygon.

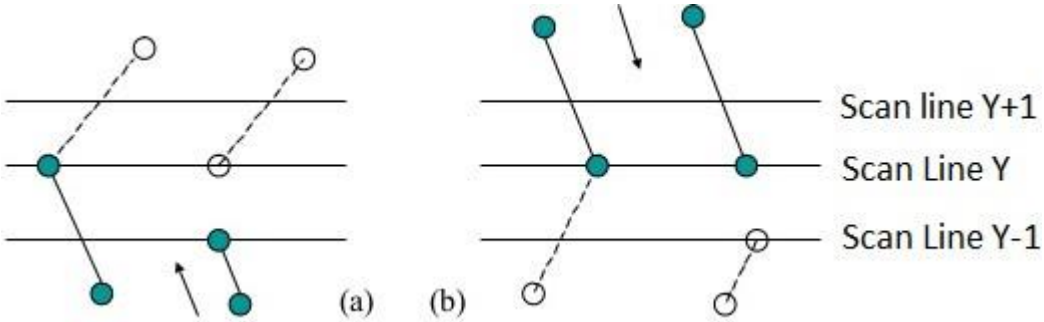


Fig. 2.18: - adjacent scan line intersects with polygon edge.

- For above figure we can write slope equation for polygon boundary as follows.
  $m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$
- Since change in y coordinates between the two scan lines is simply
  $y_{k+1} - y_k = 1$
- So slope equation can be modified as follows
  $m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$
  $m = \frac{1}{x_{k+1} - x_k}$
  $x_{k+1} - x_k = \frac{1}{m}$
  $x_{k+1} = x_k + \frac{1}{m}$
- Each successive x intercept can thus be calculated by adding the inverse of the slope and rounding to the nearest integer.
- For parallel execution of this algorithm we assign each scan line to separate processor in that case instead of using previous x values for calculation we use initial x values by using equation as.
  $x_k = x_0 + \frac{k}{m}$
- Now if we put $m = \frac{\Delta y}{\Delta x}$ in incremental calculation equation $x_{k+1} = x_k + \frac{1}{m}$ then we obtain equation as.
  $x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$

- Using this equation we can perform integer evaluation of x intercept by initializing a counter to 0, then incrementing counter by the value of Δx each time we move up to a new scan line.
- When the counter value becomes equal to or greater than Δy, we increment the current x intersection value by 1 and decrease the counter by the value Δy.
- This procedure is seen by following figure.

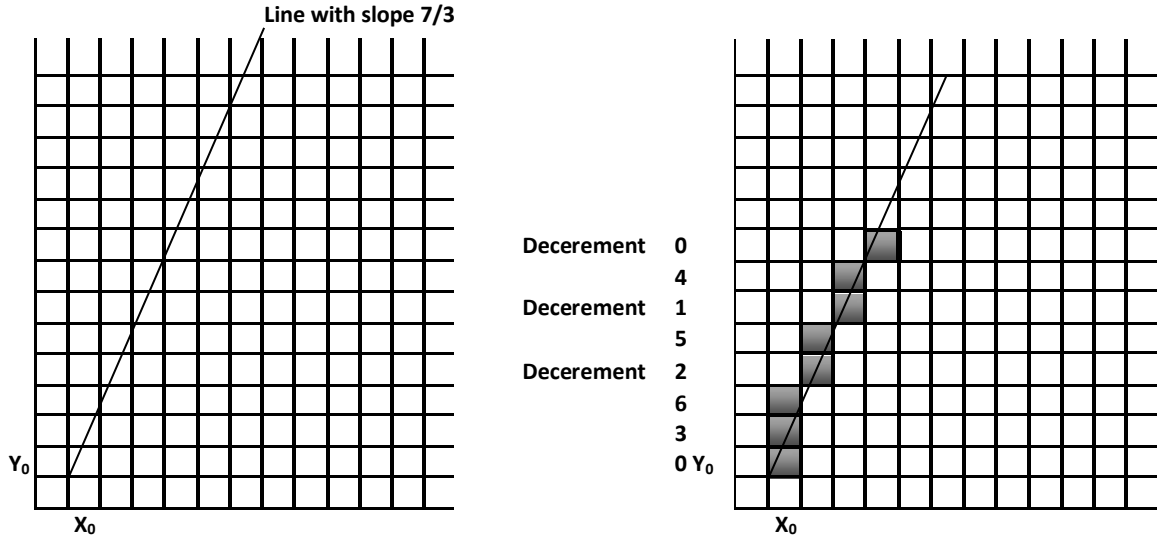


Fig. 2.19: - line with slope 7/3 and its integer calculation using equation $x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$.

- Steps for above procedure

1. Suppose m = 7/3
2. Initially, set counter to 0, and increment to 3 (which is Δx).
3. When move to next scan line, increment counter by adding Δx
4. When counter is equal or greater than 7 (which is Δy), increment the x-intercept (in other words, the x-intercept for this scan line is one more than the previous scan line), and decrement counter by 7(which is Δy).

- To efficiently perform a polygon fill, we can first store the polygon boundary in a sorted edge table that contains all the information necessary to process the scan lines efficiently.
- We use bucket sort to store the edge sorted on the smallest y value of each edge in the correct scan line positions.
- Only the non-horizontal edges are entered into the sorted edge table.
- Figure below shows one example of storing edge table.

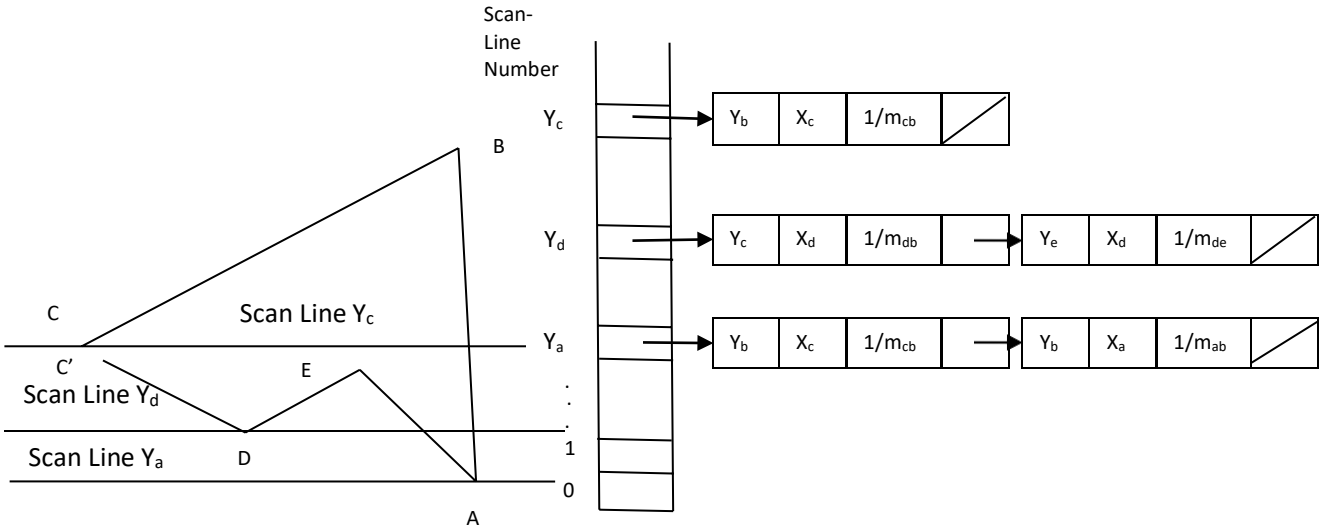


Fig. 2.20: - A polygon and its sorted edge table.

- Each entry in the table for a particular scan line contains the maximum y values for that edges, the x intercept value for edge, and the inverse slope of the edge.
- For each scan line the edges are in sorted order from left to right.
- Than we process the scan line from the bottom to top for whole polygon and produce active edge list for each scan line crossing the polygon boundaries.
- The active edge list for a scan line contains all edges crossed by that line, with iterative coherence calculation used to obtain the edge intersections.

Inside-Outside Tests

- In area filling and other graphics operation often required to find particular point is inside or outside the polygon.
- For finding which region is inside or which region is outside most graphics package use either odd even rule or the nonzero winding number rule.

Odd Even Rule

- It is also called the odd parity rule or even odd rule.
- By conceptually drawing a line from any position p to a distant point outside the coordinate extents of the object and counting the number of edges crossing by this line is odd, than p is an interior point. Otherwise p is exterior point.
- To obtain accurate edge count we must sure that line selected is does not pass from any vertices.
- This is shown in figure 2.21(a).

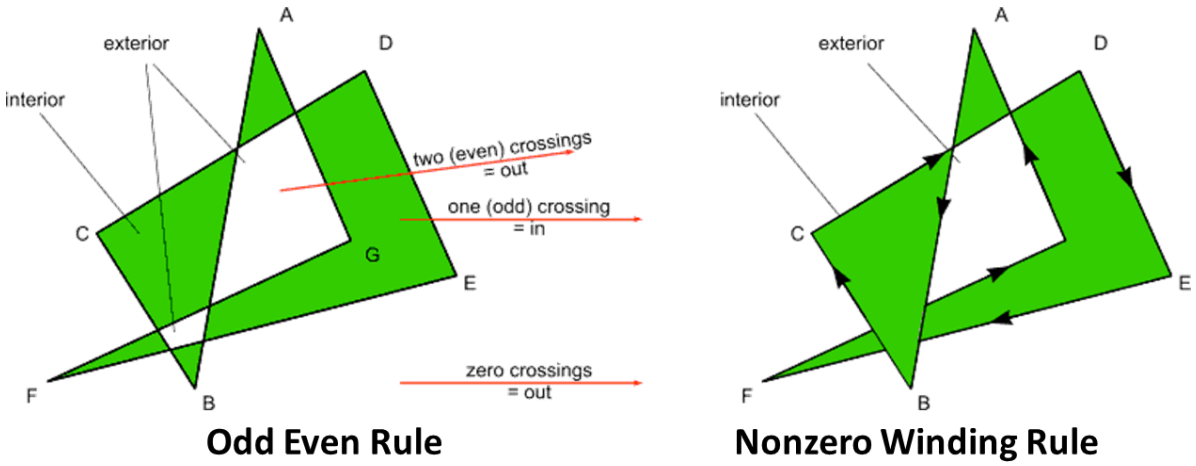


Fig. 2.21: - Identifying interior and exterior region for a self-intersecting polygon.

Nonzero Winding Number Rule

- This method counts the number of times the polygon edges wind around a particular point in the counterclockwise direction. This count is called the winding number, and the interior points of a two-dimensional object are defined to be those that have a nonzero value for the winding number.
- We apply this rule by initializing winding number with 0 and then draw a line for any point p to distant point beyond the coordinate extents of the object.
- The line we choose must not pass through vertices.
- Then we move along that line we find number of intersecting edges and we add 1 to winding number if edge cross our line from right to left and subtract 1 from winding number if edges cross from left to right.
- The final value of winding number is nonzero then the point is interior and if winding number is zero the point is exterior.
- This is shown in figure 2.21(b).
- One way to determine directional edge crossing is to take the vector cross product of a vector U along the line from p to distant point with the edge vector E for each edge that crosses the line.
- If z component of the cross product$U \times E$ for a particular edge is positive that edge is crosses from right to left and we add 1 to winding number otherwise the edge is crossing from left to right and we subtract 1 from winding number.

Comparison between Odd Even Rule and Nonzero Winding Rule

- For standard polygons and simple object both rule gives same result but for more complicated shape both rule gives different result which is illustrated in figure 2.21.

Scan-Line Fill of Curved Boundary Areas

- Scan-line fill of region with curved boundary is more time consuming as intersection calculation now involves nonlinear boundaries.
- For simple curve such as circle or ellipse scan line fill process is straight forward process.
- We calculate the two scan line intersection on opposite side of the curve.
- This is same as generating pixel position along the curve boundary using standard equation of curve.
- Then we fill the color between two boundary intersections.
- Symmetry property is used to reduce the calculation.
- Similar method can be used for fill the curve section.

Boundary Fill Algorithm/ Edge Fill Algorithm

- In this method, edges of the polygons are drawn. Then starting with some seed, any point inside the polygon we examine the neighbouring pixels to check whether the boundary pixel is reached.
- If boundary pixels are not reached, pixels are highlighted and the process is continued until boundary pixels are reached.
- Boundary defined regions may be either 4-cormected or 8-connected. as shown in the Figure below

(a) Four connected region (b) Eight connected region

Fig. 2.22: - Neighbor pixel connected to one pixel.

- If a region is 4-connected, then every pixel in the region may be reached by a combination of moves in only four directions: left, right, up and down.
- For an 8-connected region every pixel in the region may be reached by a combination of moves in the two horizontal, two vertical, and four diagonal directions.
- In some cases, an 8-connected algorithm is more accurate than the 4-connected algorithm. This is illustrated in Figure below. Here, a 4-connected algorithm produces the partial fill.

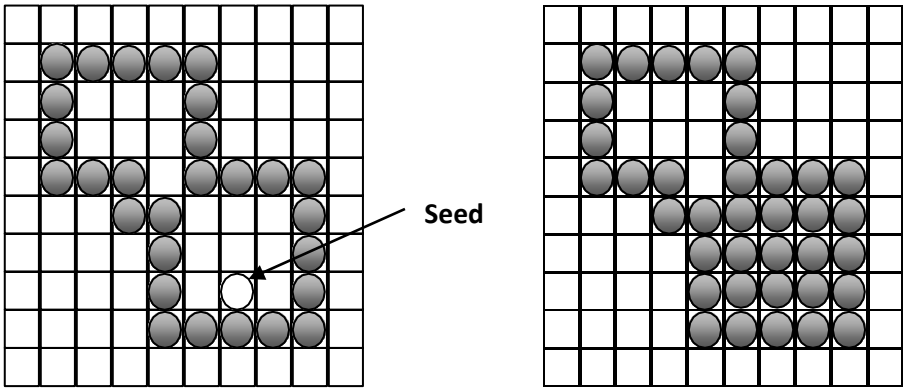


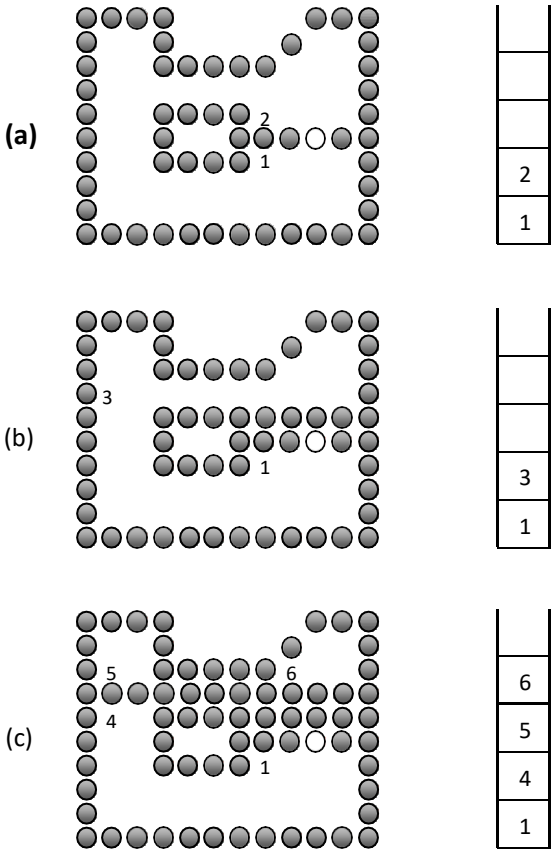
Fig. 2.23: - partial filling resulted due to using 4-connected algorithm.

- The following procedure illustrates the recursive method for filling a 4-connected region with color specified in parameter fill color (f-color) up to a boundary color specified with parameter boundary color (b-color).
- Procedure :

```
boundary-fill4(x, y, f-color, b-color)
{
    if(getpixel (x,y) ! = b-color&&gepixel (x, y) ! = f-color)
    {
            putpixel (x, y, f-color)
            boundary-fill4(x + 1, y, f-color, b-color);
```

```
            boundary-fill4(x, y + 1, f-color, b-color);
            boundary-fill4(x - 1, y, f-color, b-color);
            boundary-fill4(x, y - l, f-color, b-color);
        }
    }
```

- Note: 'getpixel' function gives the color of .specified pixel and 'putpixel' function draws the pixel with specified color.
- Same procedure can be modified according to 8 connected region algorithm by including four additional statements to test diagonal positions, such as $(x + 1, y + 1)$.
- This procedure requires considerable stacking of neighbouring points more, efficient methods are generally employed.
- This method fill horizontal pixel spans across scan lines, instead of proceeding to 4 connected or 8 connected neighbouring points.
- Then we need only stack a beginning position for each horizontal pixel span, instead of stacking all unprocessed neighbouring positions around the current position.
- Starting from the initial interior point with this method, we first fill in the contiguous span of pixels on this starting scan line.
- Then we locate and stack starting positions for spans on the adjacent scan lines, where spans are defined as the contiguous horizontal string of positions bounded by pixels displayed in the area border color.
- At each subsequent step, we unstack the next start position and repeat the process.
- An example of how pixel spans could be filled using this approach is illustrated for the 4-connected fill region in Figure below.



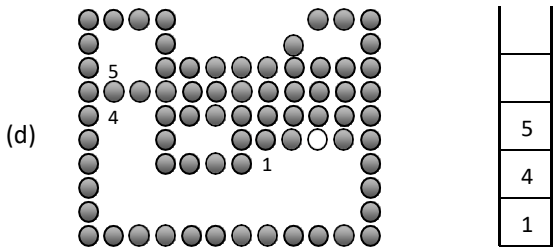


Fig. 2.24: - Boundary fill across pixel spans for a 4-connected area.

Flood-Fill Algorithm

- Sometimes it is required to fill in an area that is not defined within a single color boundary.
- In such cases we can fill areas by replacing a specified interior color instead of searching for a boundary color.
- This approach is called a flood-fill algorithm. Like boundary fill algorithm, here we start with some seed and examine the neighbouring pixels.
- However, here pixels are checked for a specified interior color instead of boundary color and they are replaced by new color.
- Using either a 4-connected or 8-connected approach, we can step through pixel positions until all interior point have been filled.
- The following procedure illustrates the recursive method for filling 4-connected region using flood-fill algorithm.
- Procedure :

```
flood-fill4(x, y, new-color, old-color)
{
    if(getpixel (x,y) = = old-color)
    {
            putpixel (x, y, new-color)
            flood-fill4 (x + 1, y, new-color, old -color);
            flood-fill4 (x, y + 1, new -color, old -color);
            flood-fill4 (x - 1, y, new -color, old -color);
            flood-fill4 (x, y - l, new -color, old-color);
    }
}
```

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

Character Generation

- We can display letters and numbers in variety of size and style.
- The overall design style for the set of character is called typeface.
- Today large numbers of typefaces are available for computer application for example Helvetica, New York platino etc.
- Originally, the term font referred to a set of cast metal character forms in a particular size and format, such as 10-point Courier Italic or 12- point Palatino Bold. Now, the terms font and typeface are often used interchangeably, since printing is no longer done with cast metal forms.
- Two different representations are used for storing computer fonts.

Bitmap Font/ Bitmapped Font

- A simple method for representing the character shapes in a particular typeface is to use rectangular grid patterns.
- Figure below shows pattern for particular letter.

1	1	1	1	1	1	0
0	1	1	0	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	1	0
0	1	1	0	0	1	1
0	1	1	0	0	1	1
1	1	1	1	1	1	0

Fig. 2.25: - Grid pattern for letter B.

- When the pattern in figure 2.25 is copied to the area of frame buffer, the 1 bits designate which pixel positions are to be displayed on the monitor.
- Bitmap fonts are the simplest to define and display as character grid only need to be mapped to a frame-buffer position.
- Bitmap fonts require more space because each variation (size and format) must be stored in a font cache.
- It is possible to generate different size and other variation from one set but this usually does not produce good result.

Outline Font

- In this method character is generated using curve section and straight line as combine assembly.
- Figure below shows how it is generated.

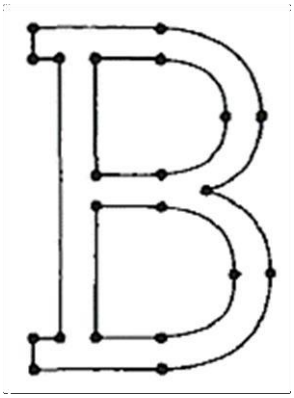


Fig. 2.26: - outline for letter B.

- To display the character shown in figure 2.26 we need to fill interior region of the character.
- This method requires less storage since each variation does not required a distinct font cache.
- We can produce boldface, italic, or different sizes by manipulating the curve definitions for the character outlines.
- But this will take more time to process the outline fonts, because they must be scan converted into the frame buffer.

Stroke Method

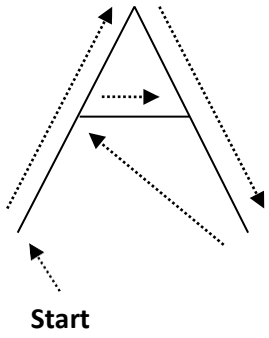


Fig. 2.27: - Stroke Method for Letter A.

- It uses small line segments to generate a character.
- The small series of line segments are drawn like a stroke of a pen to form a character as shown in figure.
- We can generate our own stroke method by calling line drawing algorithm.
- Here it is necessary to decide which line segments are needs for each character and then draw that line to display character.
- It support scaling by changing length of line segment.

Starbust Method

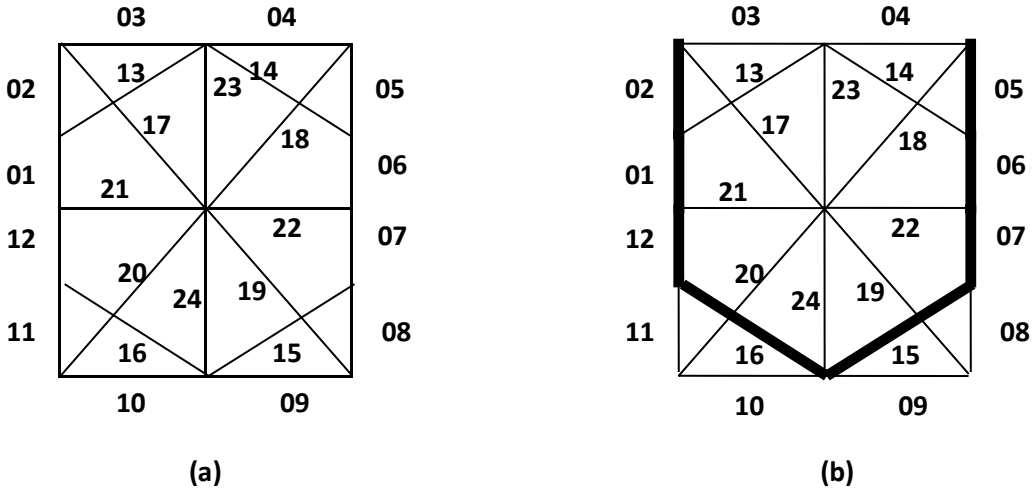


Fig. 2.28: - (a) Starbust Method. (b) Letter V using starbust method

- In this method a fix pattern of line segments are used to generate characters.
- As shown in figure 2.28 there are 24 line segments.
- We highlight those lines which are necessary to draw a particular character.
- Pattern for particular character is stored in the form of 24 bit code. In which each bit represents corresponding line having that number.
- That code contains 0 or 1 based on line segment need to highlight. We put bit value 1 for highlighted line and 0 for other line.
- Code for letter V is
  1 1 0 0 1 1 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0
- This technique is not used now a days because:
  1. It requires more memory to store 24 bit code for single character.

2. It requires conversion from code to character.
3. It doesn't provide curve shapes.

Line Attributes

- Basic attributes of a straight line segment are its type, its dimension, and its color. In some graphics packages, lines can also be displayed using selected pen or brush option.

Line Type

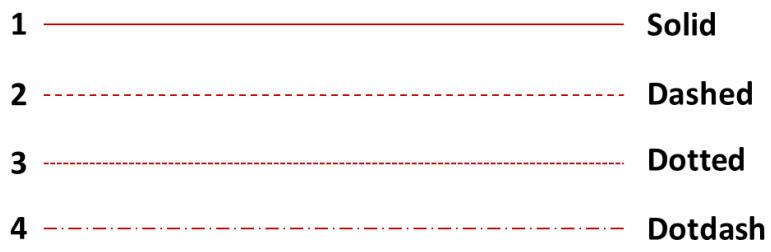


Fig. 2.29: - Line type

- Possible selection for the line-type attribute includes solid lines, dashed lines, and dotted lines etc.
- We modify a line –drawing algorithm to generate such lines by setting the length and spacing of displayed solid sections along the line path.
- A dashed line could be displayed by generating an inter dash spacing that is equal to the length of the solid sections. Both the length of the dashes and the inter dash spacing are often specified as user options.
- To set line type attributes in a PHIGS application program, a user invokes the function:
  setLinetype(lt)
- Where parameter lt is assigned a positive integer value of 1, 2, 3, 4… etc. to generate lines that are, respectively solid, dashed, dotted, or dash-dotted etc.
- Other values for the line-type parameter It could be used to display variations in the dot-dash patterns. Once the line-type parameter has been set in a PHIGS application program, all subsequent line-drawing commands produce lines with this Line type.
- Raster graphics generates these types by plotting some pixel and some pixel is off along the line path. We can generate different patterns by specifying 1 for on pixel and 0 for off pixel then we can generate 1010101 patter as a dotted line.
- It is used in many application for example comparing data in graphical form.

Line Width

- Implementation of line-width options depends on the capabilities of the output device.
- A heavy line on a video monitor could be displayed as adjacent parallel lines, while a pen plotter might require pen changes.
- To set line width attributes in a PHIGS application program, a user invokes the function:
  setLinewidthScalFactor (lw)
- Line-width parameter lw is assigned a positive number to indicate the relative width of the line to be displayed.
- Values greater than 1 produce lines thicker than the standard line width and values less than the 1 produce line thinner than the standard line width.

- In raster graphics we generate thick line by plotting above and below pixel of line path when slope $|m| < 1$ and by plotting left and right pixel of line path when slope $|m| > 1$ which is illustrated in below figure.

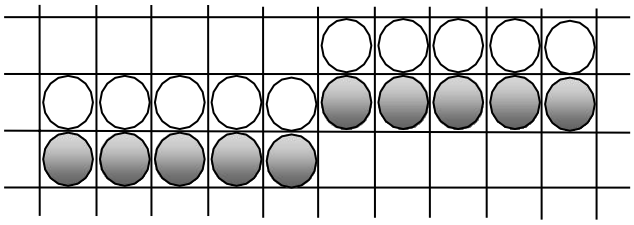


Fig. 2.30: - Double-wide raster line with slope $|m| < 1$ generated with vertical pixel spans.

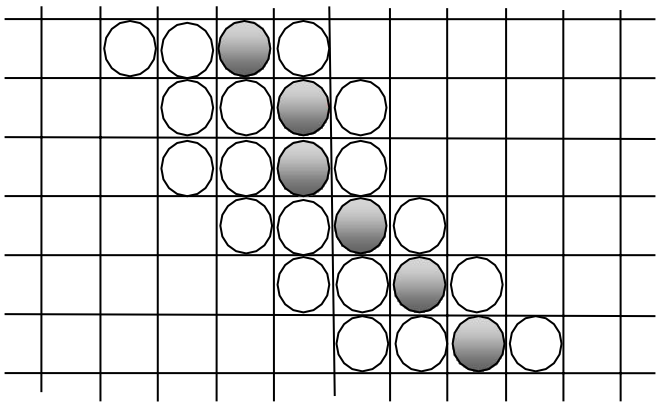


Fig. 2.31: - Raster line with slope $|m| > 1$ and line-width parameter $lw = 4$ plotted with horizontal pixel spans.

- As we change width of the line we can also change line end which are shown below which illustrate all three types of ends.

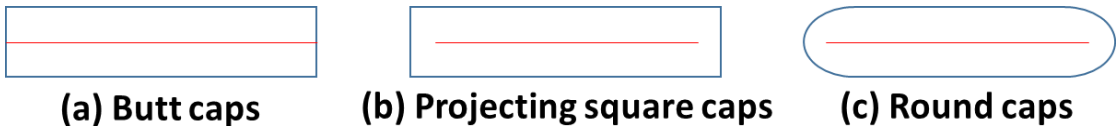


Fig. 2.32: - Thick lines drew with (a) butt caps, (b) projecting square caps, and (c) round caps.

- Similarly we generates join of two lines of modified width are shown in figure below which illustrate all three type of joins.

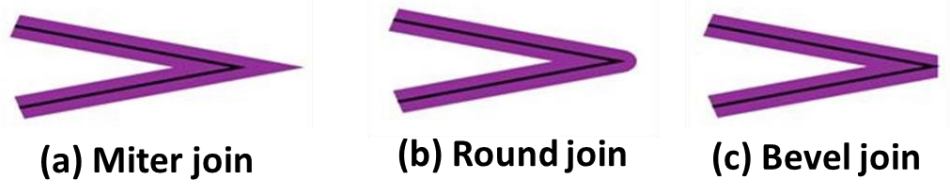


Fig. 2.33: - Thick lines segments connected with (a) miter join, (b) round join, and (c) bevel join.

Line Color

- The name itself suggests that it is defining color of line displayed on the screen.
- By default system produce line with current color but we can change this color by following function in PHIGS package as follows:
  setPolylineColorIndex (lc)
- In this lc is constant specifying particular color to be set.

Pen and Brush Options

- In some graphics packages line is displayed with pen and brush selections.
- Options in this category include shape, size, and pattern.
- Some possible pen or brush are shown in below figure.

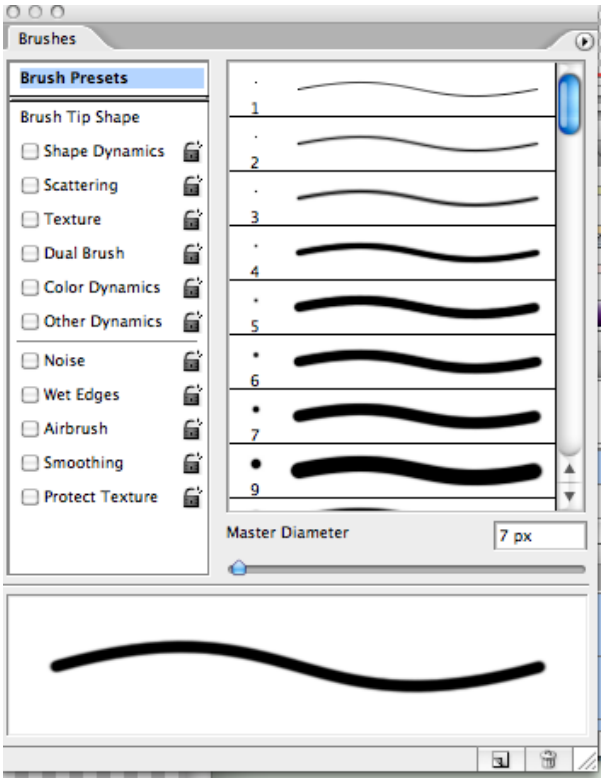


Fig. 2.34: - Pen and brush shapes for line display.

- These shapes can be stored in a pixel mask that identifies the array of pixel positions that are to be set along the line path.
- Lines generated with pen (or brush) shapes can be displayed in various widths by changing the size of the mask.
- Also, lines can be displayed with selected patterns by superimposing the pattern values onto the pen or brush mask.

Color and Greyscale Levels

- Various color and intensity-level options can be made available to a user, depending on the capabilities and design objectives of a particular system.
- General purpose raster-scan systems, for example, usually provide a wide range of colors, while random-scan monitors typically offer only a few color choices, if any.
- In a color raster system, the number of color choices available depends on the amount of storage provided per pixel in the frame buffer
- Also, color-information can be stored in the frame buffer in two ways: We can store color codes directly in the frame buffer, or we can put the color codes in a separate table and use pixel values as an index into this table
- With direct storage scheme we required large memory for frame buffer when we display more color.
- While in case of table it is reduced and we call it color table or color lookup table.

Color Lookup Table

- Color values of 24 bit is stored in lookup table and in frame buffer we store only 8 bit index which gives index of required color stored into lookup table. So that size of frame buffer is reduced and we can display more color.
- When we display picture using this technique on output screen we look into frame buffer where index of the color is stored and take 24 bit color value from look up table corresponding to frame buffer index value and display that color on particular pixel.

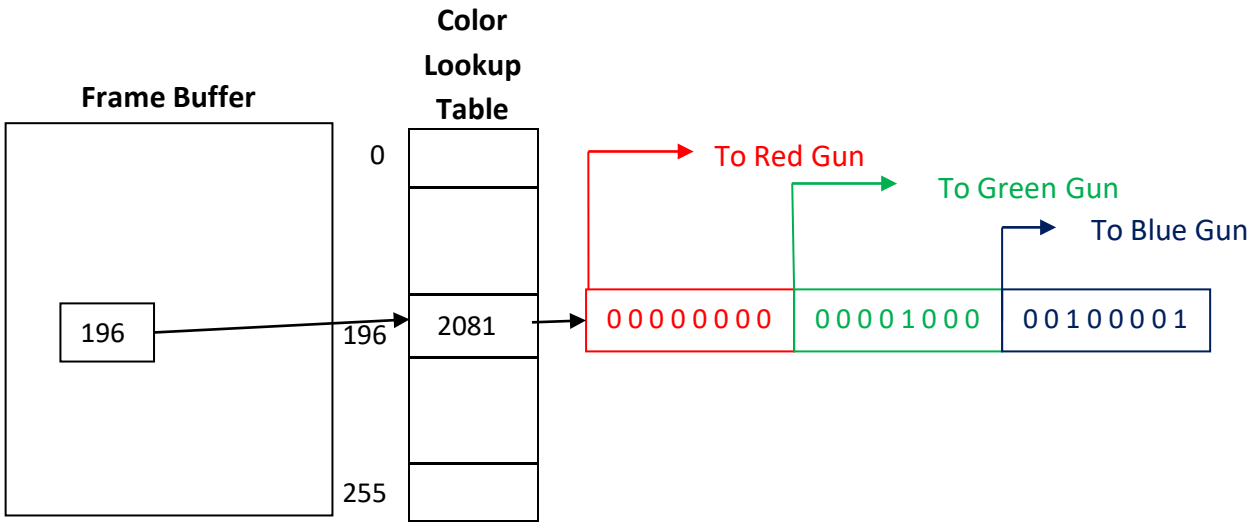


Fig. 2.35: - color lookup table.

Greyscale

- With monitors that have no color capability, color function can be used in an application program to set the shades of grey, or greyscale for display primitives.
- Numeric values between 0-to-1 can be used to specify greyscale levels.
- This numeric values is converted in binary code for store in raster system.
- Table below shows specification for intensity codes for a four level greyscale system.

Intensity Code	Stored Intensity Values In The		Displayed Greyscale
	Frame Buffer	Binary Code	
0.0	0	00	Black
0.33	1	01	Dark grey
0.67	2	10	Light grey
1.0	3	11	White

Table 2.1: - Intensity codes for a four level greyscale system.

- In this example, any intensity input value near 0.33 would be stored as the binary value 01 in the frame buffer, and pixels with this value would be displayed as dark grey.
- If more bits are available per pixel we can obtain more levels of grey scale for example with 3 bit per pixel we can achieve 8 levels of greyscale.

Area-Fill Attributes

- For filling any area we have choice between solid colors or pattern to fill all these are include in area fill attributes.
- Area can be painted by various burses and style.

Fill Styles

- Area are generally displayed with three basic style hollow with color border, filled with solid color, or filled with some design.
- In PHIGS package fill style is selected by following function.
  setInteriorStyle (fs)
- Value of fs include hollow, solid, pattern etc.
- Another values for fill style is hatch, which is patterns of line like parallel line or crossed line.
- Figure bellow shows different style of filling area.

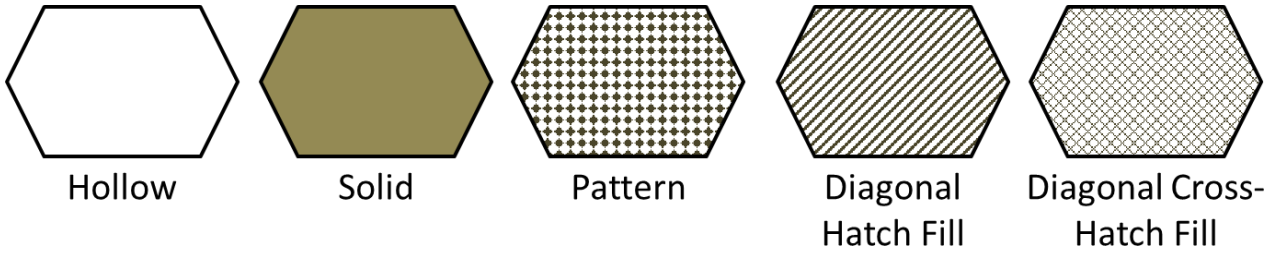


Fig. 2.36: - Different style of area filling.

- For setting interior color in PHIGS package we use:
  setInteriorColorIndex (fc)
- Where fc specify the fill color.

Pattern Fill

- We select the pattern with
  setInteriorStyleIndex (pi)
- Where pattern index parameter pi specifies position in pattern table entry.
- Figure below shows pattern table.

Index(pi)	Pattern(cp)
1	$\begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$
2	$\begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix}$

Table 2.2: - Pattern table.

- For example, the following set of statements would fill the area defined in the fillArea command with the second pattern type stored in the pattern table:
  SetInteriorStyle(pattern) ;
  setInteriorStyleIndex (2) ;
  fillArea (n, points);
- Separate table can be maintain for hatch pattern and we can generate our own table with required pattern.
- Other function used for setting other style as follows
  setpatternsize (dx, dy)
  setPaternReferencePoint (positicn)
- We can create our own pattern by setting and resetting group of pixel and then map it into the color matrix.

Soft Fill

- Soft fill is modified boundary fill and flood fill algorithm in which we are fill layer of color on back ground color so that we can obtain the combination of both color.
- It is used to recolor or repaint so that we can obtain layer of multiple color and get new color combination.
- One use of this algorithm is soften the fill at boundary so that blurred effect will reduce the aliasing effect.
- For example if we fill t amount of foreground color then pixel color is obtain as:
  $p = tF + (1 - t)B$
- Where F is foreground color and B is background color
- If we see this color in RGB component then:
  $p = (p_r, p_g, p_b)$ $f = (f_r, f_g, f_b)$ $b = (b_r, b_g, b_b)$
- Then we can calculate t as follows:
  $t = \frac{P_k - B_k}{F_k - B_k}$
- If we use more than two color say three at that time equation becomes as follow:
  $p = t_0F + t_1B_1 + (1 - t_0 - t_1)B_2$
- Where the sum of coefficient t_0, t_1, and $(1 - t_0 - t_1)$ is 1.

Character Attributes

- The appearance of displayed characters is controlled by attributes such as font, size, color, and orientation.
- Attributes can be set for entire string or may be individually.

Text Attributes

- In text we are having so many style and design like italic fonts, bold fonts etc.
- For setting the font style in PHIGS package we have one function which is:
  setTextFont (tf)
- Where tf is used to specify text font
- It will set specified font as a current character.
- For setting color of character in PHIGS we have function:
  setTextColorIndex (tc)
- Where text color parameter tc specifies an allowable color code.
- For setting the size of the text we use function.
  setCharacterheight (ch)
- For scaling the character we use function.
  setCharacterExpansionFacter (cw)
- Where character width parameter cw is set to a positive real number that scale the character body width.
- Spacing between character is controlled by function
  setCharacterSpacing (cs)
- Where character spacing parameter cs can be assigned any real value.
- The orientation for a displayed character string is set according to the direction of the character up vector:
  setCharacterUpVector (upvect)
- Parameter upvect in this function is assigned two values that specify the x and y vector components.

- Text is then displayed so that the orientation of characters from baseline to cap line is in the direction of the up vector.
- For setting the path of the character we use function:
  setTextPath (tp)
- Where the text path parameter tp can be assigned the value: right, left, up, or down.
- It will set the direction in which we are writing.
- For setting the alignment of the text we use function.
  setTextAlignment (h, v)
- Where parameter h and v control horizontal and vertical alignment respectively.
- For specifying precision for text display is given with function.
  setTextPrecision (tpr)
- Where text precision parameter tpr is assigned one of the values: string, char, or stroke.
- The highest-quality text is produced when the parameter is set to the value stroke.

Marker Attributes

- A marker symbol display single character in different color and in different sizes.
- For marker attributes implementation by procedure that load the chosen character into the raster at defined position with the specified color and size.
- We select marker type using function.
  setMarkerType (mt)
- Where marker type parameter mt is set to an integer code.
- Typical codes for marker type are the integers 1 through 5, specifying, respectively, a dot (.), a vertical cross (+), an asterisk (*), a circle (o), and a diagonal cross (x). Displayed marker types are centred on the marker coordinates.
- We set the marker size with function.
  SetMarkerSizeScaleFactor (ms)
- Where parameter marker size ms assigned a positive number according to need for scaling.
- For setting marker color we use function.
  setPolymarkerColorIndex (mc)
- Where parameter mc specify the color of the marker symbol.

UNIT - II

***Prepared By: GM SUBHANI*, Asst.Professor**
MALLAREDDY INSTITUTE OF TECHNOLOGY & SCIENCE
Maisammaguda, Secunderabad, Telangana.

III YR – I SEM CSE & IT

Transformation

Changing Position, shape, size, or orientation of an object on display is known as transformation.

Basic Transformation

- Basic transformation includes three transformations **Translation**, **Rotation**, and **Scaling**.
- These three transformations are known as basic transformation because with combination of these three transformations we can obtain any transformation.

Translation

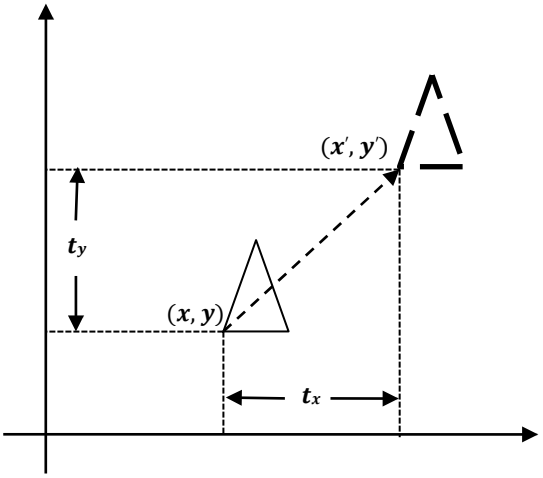


Fig. 3.1: - Translation.

- It is a transformation that used to reposition the object along the straight line path from one coordinate location to another.
- It is rigid body transformation so we need to translate whole object.
- We translate two dimensional point by adding translation distance t_x and t_y to the original coordinate position (x, y) to move at new position (x', y') as:
  $x' = x + t_x$ **&** $y' = y + t_y$
- Translation distance pair (t_x, t_y) is called a **Translation Vector** or **ShiftVector**.
- We can represent it into single matrix equation in column vector as;
  $P' = P + T$

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

- We can also represent it in row vector form as:
  $P' = P + T$
  $[x' \quad y'] = [x \quad y] + [t_x \quad t_y]$
- Since column vector representation is standard mathematical notation and since many graphics package like **GKS** and **PHIGS** uses column vector we will also follow column vectorrepresentation.
- **Example**: - Translate the triangle [A (10, 10), B (15, 15), C (20, 10)] 2 unit in x direction and 1 unit in y direction.
  We know that
  $P' = P + T$

$$P' = [P] + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

For point (10, 10)

$$A' = \begin{bmatrix} 10 \\ 10 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$A' = \begin{bmatrix} 12 \\ 11 \end{bmatrix}$$

For point (15, 15)

$$B' = \begin{bmatrix} 15 \\ 15 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$B' = \begin{bmatrix} 17 \\ 16 \end{bmatrix}$$

For point (10, 10)

$$C' = \begin{bmatrix} 20 \\ 10 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$C' = \begin{bmatrix} 22 \\ 11 \end{bmatrix}$$

- Final coordinates after translation are [A′ (12, 11), B′ (17, 16), C′(22,11)].

Rotation

- It is a transformation that used to reposition the object along the circular path in the XY - plane.
- To generate a rotation we specify a rotation angle $\boldsymbol{\theta}$ and the position of the **Rotation Point** (**Pivot Point**) $(\boldsymbol{x_r}, \boldsymbol{y_r})$ about which the object is to be rotated.
- Positive value of rotation angle defines counter clockwise rotation and negative value of rotation angle defines clockwise rotation.
- We first find the equation of rotation when pivot point is at coordinate origin $(\mathbf{0},\mathbf{0})$.

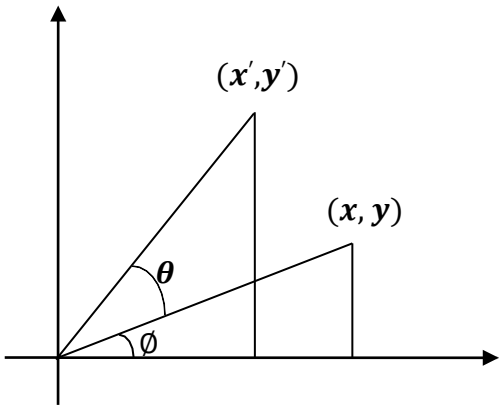


Fig. 3.2: - Rotation.

- From figure we can write.

  $\boldsymbol{x = r \cos \emptyset}$

  $\boldsymbol{y = r \sin \emptyset}$

  and

  $\boldsymbol{x' = r \cos(\theta + \emptyset) = r \cos \emptyset \cos \theta - r \sin \emptyset \sin \theta}$

  $\boldsymbol{y' = r \sin(\emptyset + \theta) = r \cos \emptyset \sin \theta + r \sin \emptyset \cos \theta}$

- Now replace $r\cos\emptyset$ with x and $r\sin\emptyset$ with y in above equation.
  $x' = x\cos\theta - y\sin\theta$
  $y' = x\sin\theta + y\cos\theta$
- We can write it in the form of column vector matrix equation as;
  $P' = R \cdot P$

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

- Rotation about arbitrary point is illustrated in below figure.

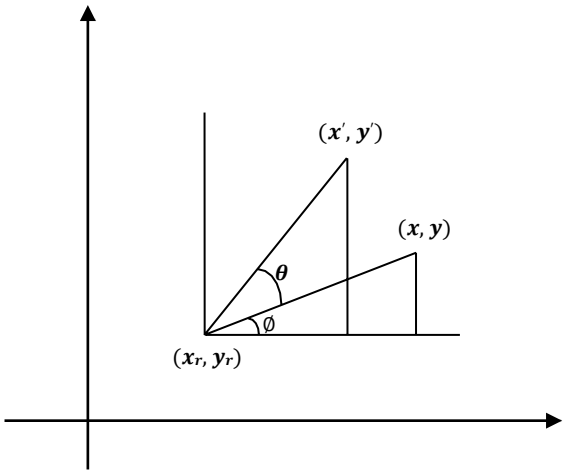


Fig. 3.3: - Rotation about pivot point.

- Transformation equation for rotation of a point about pivot point (x_r, y_r) is:
  $x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$
  $y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$
- These equations are differing from rotation about origin and its matrix representation is also different.
- Its matrix equation can be obtained by simple method that we will discuss later in this chapter.
- Rotation is also rigid body transformation so we need to rotate each point of object.
- **Example**: - Locate the new position of the triangle [A (5, 4), B (8, 3), C (8, 8)] after its rotation by 90° clockwise about the origin.
  As rotation is clockwise we will take $\theta = -90^{\circ}$.
  $P' = R \cdot P$

$$P' = \begin{bmatrix} \cos(-90) & -\sin(-90) \\ \sin(-90) & \cos(-90) \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \end{bmatrix}$$
$$P' = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \end{bmatrix}$$
$$P' = \begin{bmatrix} 4 & 3 & 8 \\ -5 & -8 & -8 \end{bmatrix}$$

- Final coordinates after rotation are [A′ (4, -5), B′ (3, -8), C′ (8, -8)].

Scaling

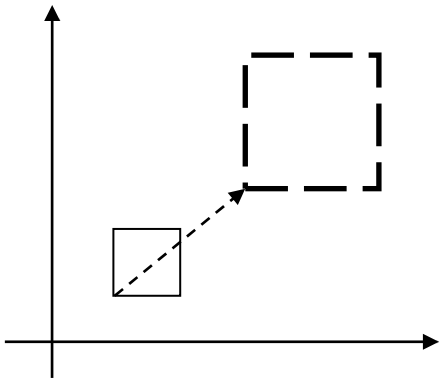


Fig. 3.4: - Scaling.

It is a transformation that used to alter the size of an object.

- This operation is carried out by multiplying coordinate value $(\boldsymbol{x}, \boldsymbol{y})$ with scaling factor $(\boldsymbol{s_x}, \boldsymbol{s_y})$ respectively.
- So equation for scaling is given by:
  $\boldsymbol{x'} = \boldsymbol{x} \cdot \boldsymbol{s_x}$
  $\boldsymbol{y'} = \boldsymbol{y} \cdot \boldsymbol{s_y}$
- These equation can be represented in column vector matrix equation as:
  $\boldsymbol{P'} = \boldsymbol{S} \cdot \boldsymbol{P}$

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

- Any positive value can be assigned to $(\boldsymbol{s_x}, \boldsymbol{s_y})$.
- Values less than 1 reduce the size while values greater than 1 enlarge the size of object, and object remains unchanged when values of both factor is 1.
- Same values of $\boldsymbol{s_x}$ and $\boldsymbol{s_y}$ will produce **Uniform Scaling**. And different values of $\boldsymbol{s_x}$ and $\boldsymbol{s_y}$ will produce **Differential Scaling**.
- Objects transformed with above equation are both scale and repositioned.
- Scaling factor with value less than 1 will move object closer to origin, while scaling factor with value greater than 1 will move object away from origin.
- We can control the position of object after scaling by keeping one position fixed called **Fix point** $(\boldsymbol{x_f}, \boldsymbol{y_f})$ that point will remain unchanged after the scaling transformation.

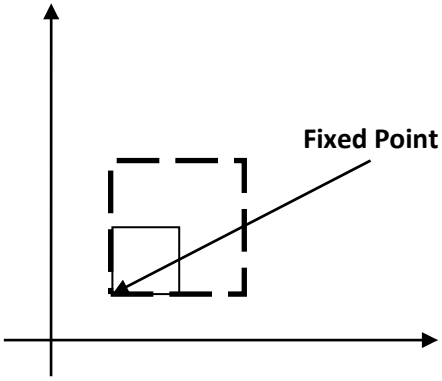


Fig. 3.5: - Fixed point scaling.

- Equation for scaling with fixed point position as (x_f, y_f) is:

$x' = x_f + (x - x_f)s_x$ $\quad$ $y' = y_f + (y - y_f)s_y$

$x' = x_f + xs_x - x_f s_x$ $\quad$ $y' = y_f + ys_y - y_f s_y$

$x' = xs_x + x_f(1 - s_x)$ $\quad$ $y' = ys_y + y_f(1 - s_y)$

- Matrix equation for the same will discuss in later section.
- Polygons are scaled by applying scaling at coordinates and redrawing while other body like circle and ellipse will scale using its defining parameters. For example ellipse will scale using its semi major axis, semi minor axis and center point scaling and redrawing at that position.
- **Example**: - Consider square with left-bottom corner at (2, 2) and right-top corner at (6, 6) apply the transformation which makes its size half.

  As we want size half so value of scale factor are $s_x = 0.5$, $s_y = 0.5$ and Coordinates of square are [A (2, 2), B (6, 2), C (6, 6), D (2,6)].

$$P' = S \cdot P$$

$$P' = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \end{bmatrix}$$

$$P' = \begin{bmatrix} 1 & 3 & 3 & 1 \\ 1 & 1 & 3 & 3 \end{bmatrix}$$

- Final coordinate after scaling are [A′(1, 1), B′ (3, 1), C′ (3, 3), D′ (1,3)].

Matrix Representation and homogeneous coordinates

- Many graphics application involves sequence of geometric transformations.
- For example in design and picture construction application we perform Translation, Rotation, and scaling to fit the picture components into their proper positions.
- For efficient processing we will reformulate transformation sequences.
- We have matrix representation of basic transformation and we can express it in the general matrix form as:

  $\boldsymbol{P' = M_1 \cdot P + M_2}$

  Where $\boldsymbol{P}$ and $\boldsymbol{P'}$ are initial and final point position, $\boldsymbol{M_1}$ contains rotation and scaling terms and $\boldsymbol{M_2}$ contains translation al terms associated with pivot point, fixed point and reposition.
- For efficient utilization we must calculate all sequence of transformation in one step and for that reason we reformulate above equation to eliminate the matrix addition associated with translation terms in matrix $\boldsymbol{M_2}$.
- We can combine that thing by expanding 2X2 matrix representation into 3X3 matrices.
- It will allows us to convert all transformation into matrix multiplication but we need to represent vertex position (x, y) with homogeneous coordinate triple (x_h, y_h, h) Where $x = \frac{x_h}{h}$, $y = \frac{y_h}{h}$ thus we can also write triple as $(h \cdot x, h \cdot y, h)$.
- For two dimensional geometric transformation we can take value of h is any positive number so we can get infinite homogeneous representation for coordinate value (x,y).
- But convenient choice is set $h = 1$ as it is multiplicative identity, than (x, y) is represented as $(x, y,1)$.
- Expressing coordinates in homogeneous coordinates form allows us to represent all geometric transformation equations as matrix multiplication.
- Let's see each representation with $h = 1$

Translation

$$P' = T_{(t_x,t_y)} \cdot P$$

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

NOTE: - Inverse of translation matrix is obtain by putting $-t_x$ & $-t_y$ instead of t_x & t_y.

Rotation

$$P' = R_{(\theta)} \cdot P$$

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

NOTE: - Inverse of rotation matrix is obtained by replacing θ by $-\theta$.

Scaling

$$P' = S_{(s_x,s_y)} \cdot P$$

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

NOTE: - Inverse of scaling matrix is obtained by replacing s_x & s_y by $\frac{1}{s_x}$ & $\frac{1}{s_y}$ respectively.

Composite Transformation

- We can set up a matrix for any sequence of transformations as a **composite transformation matrix** by calculating the matrix product of individual transformation.
- For column matrix representation of coordinate positions, we form composite transformations by multiplying matrices in order from right to left.

Translations

- Two successive translations are performed as:

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

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

$$P' = \begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

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

  Here P' and P are column vector of final and initial point coordinate respectively.

- This concept can be extended for any number of successive translations.

Example: Obtain the final coordinates after two translations on point $p(2,3)$ with translation vector $(4, 3)$ and $(-1, 2)$ respectively.

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

$$P' = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot P = \begin{bmatrix} 1 & 0 & 4 + (-1) \\ 0 & 1 & 3 + 2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} \qquad P' = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \\ 1 \end{bmatrix}$$

Final Coordinates after translations are $p'(5, 8)$.

Rotations

- Two successive Rotations are performed as:

$$P' = R(\theta_2) \cdot \{R(\theta_1) \cdot P\}$$
$$P' = \{R(\theta_2) \cdot R(\theta_1)\} \cdot P$$
$$P' = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$
$$P' = \begin{bmatrix} \cos\theta_2\cos\theta_1 - \sin\theta_2\sin\theta_1 & -\sin\theta_1\cos\theta_2 - \sin\theta_2\cos\theta_1 & 0 \\ \sin\theta_1\cos\theta_2 + \sin\theta_2\cos\theta_1 & \cos\theta_2\cos\theta_1 - \sin\theta_2\sin\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = \begin{bmatrix} \cos(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) & 0 \\ \sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$
$$P' = R(\theta_1 + \theta_2) \cdot P$$

  Here P' and P are column vector of final and initial point coordinate respectively.

- This concept can be extended for any number of successive rotations.

Example: Obtain the final coordinates after two rotations on point $p(6,9)$ with rotation angles are 30^o and 60^o respectively.

$$P' = R(\theta_1 + \theta_2) \cdot P$$
$$P' = \begin{bmatrix} cos(\theta_1 + \theta_2) & -sin(\theta_1 + \theta_2) & 0 \\ sin(\theta_1 + \theta_2) & cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$
$$P' = \begin{bmatrix} cos(30 + 60) & -sin(30 + 60) & 0 \\ sin(30 + 60) & cos(30 + 60) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$
$$P' = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 6 \\ 9 \\ 1 \end{bmatrix} = \begin{bmatrix} -9 \\ 6 \\ 1 \end{bmatrix}$$

Final Coordinates after rotations are $p'(-9, 6)$.

Scaling

- Two successive scaling are performed as:

$$P' = S(s_{x2}, s_{y2}) \cdot \{S(s_{x1}, s_{y1}) \cdot P\}$$
$$P' = \{S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})\} \cdot P$$
$$P' = \begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$
$$P' = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$
$$P' = S(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2}) \cdot P$$

  Here P' and P are column vector of final and initial point coordinate respectively.

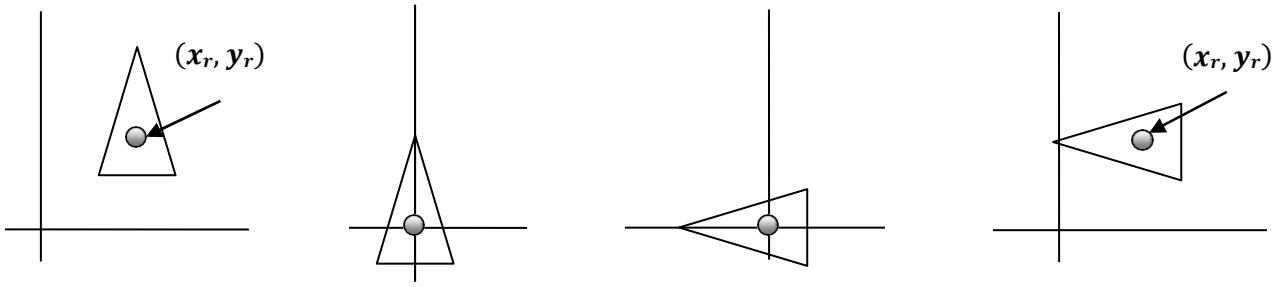
- This concept can be extended for any number of successive scaling.

Example: Obtain the final coordinates after two scaling on line pq $[p(2,2), q(8, 8)]$ with scaling factors are $(2, 2)$ and $(3, 3)$ respectively.

$$P' = S(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2}) \cdot P$$
$$P' = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P = \begin{bmatrix} 2 \cdot 3 & 0 & 0 \\ 0 & 2 \cdot 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$
$$P' = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 8 \\ 2 & 8 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 12 & 48 \\ 12 & 48 \\ 1 & 1 \end{bmatrix}$$

Final Coordinates after rotations are $p'(12, 12)$ and $q'(48, 48)$.

General Pivot-Point Rotation



(a) Original Position of Object and Pivot Point.

(b) Translation of Object so that Pivot Point (x_r, y_r) is at Origin.

(c) Rotation about Origin.

(d) Translation of Object so that Pivot Point is Return to Position (x_r, y_r) .

Fig. 3.6: - General pivot point rotation.

- For rotating object about arbitrary point called pivot point we need to apply following sequence of transformation.
  1. Translate the object so that the pivot-point coincides with the coordinate origin.
  2. Rotate the object about the coordinate origin with specified angle.
  3. Translate the object so that the pivot-point is returned to its original position (i.e. Inverse of step-1).
- Let's find matrix equation for this

$$P' = T(x_r, y_r) \cdot [R(\theta) \cdot \{T(-x_r, -y_r) \cdot P\}]$$
$$P' = \{T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r)\} \cdot P$$
$$P' = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$
$$P' = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$
$$P' = R(x_r, y_r\theta) \cdot P$$

Here P' and P are column vector of final and initial point coordinate respectively and (x_r, y_r) are the coordinates of pivot-point.

- **Example**: - Locate the new position of the triangle [A (5, 4), B (8, 3), C (8, 8)] after its rotation by 90° clockwise about the centroid.
  Pivot point is centroid of the triangle so:

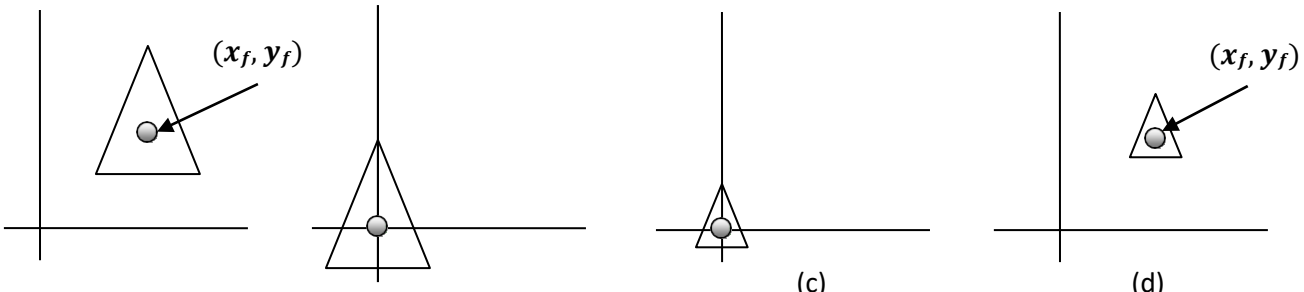
$$x_r = \frac{5+8+8}{3} = 7, \qquad y_r = \frac{4+3+8}{3} = 5$$

As rotation is clockwise we will take $\theta = -90^\circ$.

$$P' = R_{(x_r, y_r, \theta)} \cdot P$$
$$P' = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$
$$P' = \begin{bmatrix} \cos(-90) & -\sin(-90) & 7(1-\cos(-90)) + 5\sin(-90) \\ \sin(-90) & \cos(-90) & 5(1-\cos(-90)) - 7\sin(-90) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$
$$P' = \begin{bmatrix} 0 & 1 & 7(1-0) - 5(1) \\ -1 & 0 & 5(1-0) + 7(1) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$
$$P' = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 12 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$

- Final coordinates after rotation are [A′ (11, 7), B′ (13, 4), C′ (18,4)].

General Fixed-Point Scaling



(a) Original Position of Object and Fixed Point

(b) Translate Object so that Fixed Point (x_f, y_f) is at Origin

(c) Scale Object with Respect to Origin

(d) Translate Object so that Fixed Point is Return to Position (x_f, y_f) .

Fig. 3.7: - General fixed point scaling.

- For scaling object with position of one point called fixed point will remains same, we need to apply following sequence of transformation.
  1. Translate the object so that the fixed-point coincides with the coordinate origin.
  2. Scale the object with respect to the coordinate origin with specified scale factors.
  3. Translate the object so that the fixed-point is returned to its original position (i.e. Inverse of step-1).
- Let's find matrix equation for this

$$P' = T(x_f, y_f) \cdot [S(s_x, s_y) \cdot \{T(-x_f, -y_f) \cdot P\}]$$

$$P' = \{T(x_f, y_f) \cdot S(s_x, s_y) \cdot T(-x_f, -y_f)\} \cdot P$$

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

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

$$P' = S(x_f, y_f, s_x, s_y) \cdot P$$

Here P' and P are column vector of final and initial point coordinate respectively and (x_f, y_f) are the coordinates of fixed-point.

- **Example**: - Consider square with left-bottom corner at (2, 2) and right-top corner at (6, 6) apply the transformation which makes its size half such that its center remains same.
  Fixed point is center of square so:

$$x_f = 2 + \frac{6-2}{2}, \quad y_f = 2 + \frac{6-2}{2}$$

As we want size half so value of scale factor are $s_x = 0.5$, $s_y = 0.5$ and Coordinates of square are [A (2, 2), B (6, 2), C (6, 6), D (2, 6)].

$$P' = S(x_f, y_f, s_x, s_y) \cdot P$$

$$P' = \begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.5 & 0 & 4(1-0.5) \\ 0 & 0.5 & 4(1-0.5) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.5 & 0 & 2 \\ 0 & 0.5 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 3 & 5 & 5 & 3 \\ 3 & 3 & 5 & 5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

- Final coordinate after scaling are [A′(3, 3), B′ (5, 3), C′ (5, 5), D′ (3, 5)]

General Scaling Directions

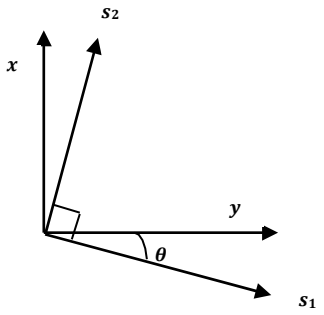


Fig. 3.8: - General scaling direction.

- Parameter s_x and s_y scale the object along x and y directions. We can scale an object in other directions by rotating the object to align the desired scaling directions with the coordinate axes before applying the scaling transformation.
- Suppose we apply scaling factor s_1 and s_2 in direction shown in figure than we will apply following transformations.
  1. Perform a rotation so that the direction for s_1 and s_2 coincide with x and y axes.
  2. Scale the object with specified scale factors.
  3. Perform opposite rotation to return points to their original orientations. (i.e. Inverse of step-1).
- Let's find matrix equation for this

$$P' = R^{-1}(\theta) \cdot [S(s_1, s_2) \cdot \{R(\theta) \cdot P\}]$$

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

$$P' = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

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

Here P' and P are column vector of final and initial point coordinate respectively and θ is the angle between actual scaling direction and our standard coordinate axes.

Other Transformation

- Some package provides few additional transformations which are useful in certain applications. Two such transformations are reflection and shear.

Reflection

- A reflection is a transformation that produces a mirror image of an object.
- The mirror image for a two –dimensional reflection is generated relative to an **axis of reflection** by rotating the object 180° about the reflection axis.
- Reflection gives image based on position of axis of reflection. Transformation matrix for few positions are discussed here.

Transformation matrix for reflection about the line $y = 0$, ***the x axis***.

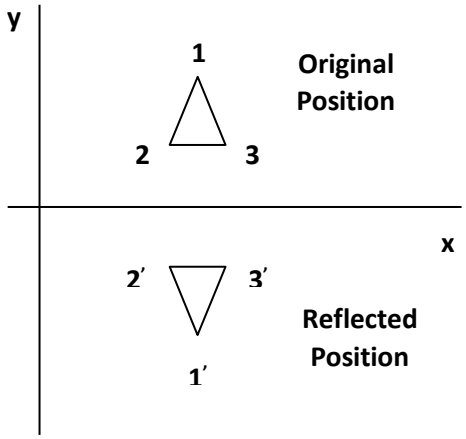


Fig. 3.9: - Reflection about x - axis.

- This transformation keeps x values are same, but flips (Change the sign) y values of coordinate positions.

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

Transformation matrix for reflection about the line $x = 0$, ***the y axis***.

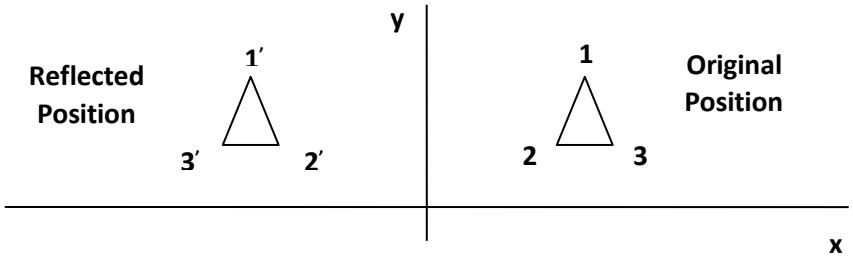


Fig. 3.10: - Reflection about y - axis.

- This transformation keeps y values are same, but flips (Change the sign) x values of coordinate positions.

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

Transformation matrix for reflection about the ***Origin***.

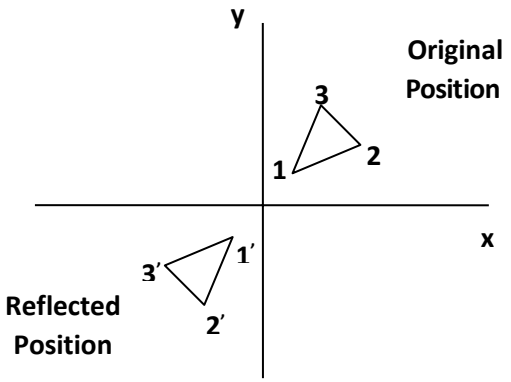


Fig. 3.11: - Reflection about origin.

- This transformation flips (Change the sign) x and y both values of coordinate positions.

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

Transformation matrix for reflection about the line $x = y$.

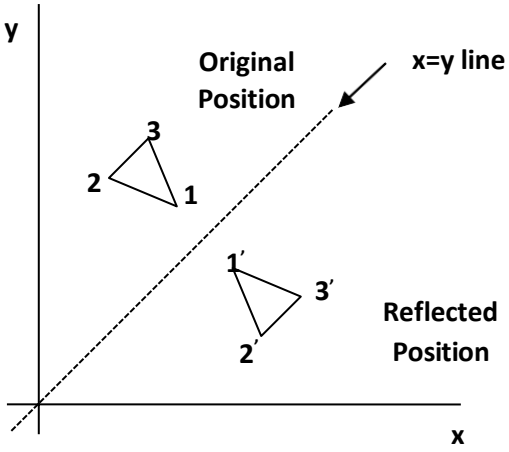


Fig. 3.12: - Reflection about x=y line.

- This transformation interchange x and y values of coordinate positions.

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

Transformation matrix for reflection about the line $x = -y$.

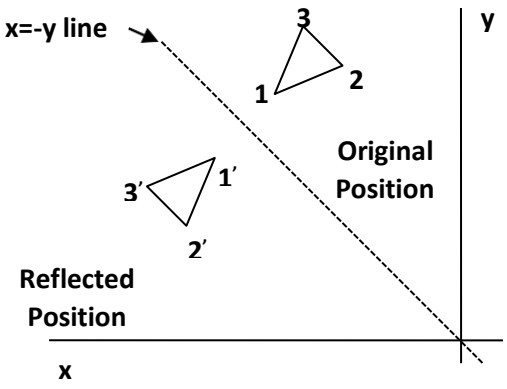


Fig. 3.12: - Reflection about x=-y line.

- This transformation interchange x and y values of coordinate positions.

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

- **Example**: - Find the coordinates after reflection of the triangle [A (10, 10), B (15, 15), C (20, 10)] about x axis.

$$P' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 & 15 & 20 \\ 10 & 15 & 10 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 10 & 15 & 20 \\ -10 & -15 & -10 \\ 1 & 1 & 1 \end{bmatrix}$$

- Final coordinate after reflection are [A′ (10, -10), B′ (15, -15), C′ (20, -10)]

Shear

- A transformation that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called **shear**.
- Two common shearing transformations are those that shift coordinate x values and those that shift y values.

Shear in $x - direction$.

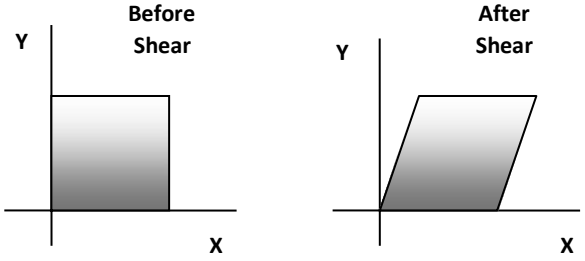


Fig. 3.13: - Shear in x-direction.

- Shear relative to $x - axis$ that is $y = 0$ line can be produced by following equation:
  $\boldsymbol{x' = x + sh_x \cdot y}$, $\boldsymbol{y' = y}$
- Transformation matrix for that is:
  $$\begin{bmatrix} \mathbf{1} & \boldsymbol{sh_x} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$
  Here $\boldsymbol{sh_x}$ is shear parameter. We can assign any real value to $\boldsymbol{sh_x}$.
- We can generate $x - direction$ shear relative to other reference line $y = y_{ref}$ with following equation:
  $\boldsymbol{x' = x + sh_x \cdot (y - y_{ref})}$, $\boldsymbol{y' = y}$
- Transformation matrix for that is:
  $$\begin{bmatrix} \mathbf{1} & \boldsymbol{sh_x} & \boldsymbol{-sh_x \cdot y_{ref}} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$
- **Example**: - Shear the unit square in x direction with shear parameter ½ relative to line $y = -1$.
  Here $y_{ref} = -1$ and $sh_x = 0.5$
  Coordinates of unit square are [A (0, 0), B (1, 0), C (1, 1), D (0, 1)].
- Final coordinate after shear are [A′ (0.5, 0), B′ (1.5, 0), C′ (2, 1), D′ (1, 1)]

$$P' = \begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
$$P' = \begin{bmatrix} 1 & 0.5 & -0.5 \cdot (-1) \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
$$P' = \begin{bmatrix} 1 & 0.5 & 0.5 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
$$P' = \begin{bmatrix} 0.5 & 1.5 & 2 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

- Shear in $\boldsymbol{y - direction}$.

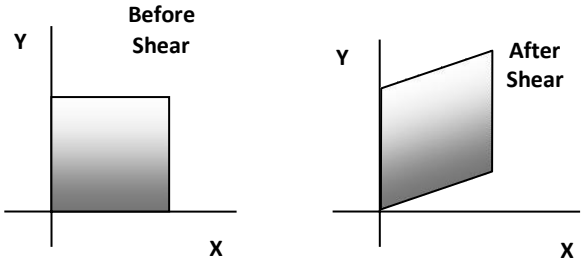


Fig. 3.14: - Shear in y-direction.

- Shear relative to $y - axis$ that is $x = 0$ line can be produced by following equation:
  $\boldsymbol{x' = x}$, $\boldsymbol{y' = y + sh_y \cdot x}$
- Transformation matrix for that is:
  $$\begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{sh_y} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$
  Here $\boldsymbol{sh_y}$ is shear parameter. We can assign any real value to $\boldsymbol{sh_y}$.

- We can generate $y - direction$ shear relative to other reference line $x = x_{ref}$ with following equation:
  $\boldsymbol{x' = x}, \qquad \boldsymbol{y' = y + sh_y \cdot (x - x_{ref})}$
- Transformation matrix for that is:

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

- **Example**: - Shear the unit square in y direction with shear parameter ½ relative to line $x = -1$.
  Here $x_{ref} = -1$ and $sh_y = 0.5$
  Coordinates of unit square are [A (0, 0), B (1, 0), C (1, 1), D (0, 1)].

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

$$P' = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & -0.5 \cdot (-1) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0.5 & 1 & 2 & 1.5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

- Final coordinate after shear are [A′(0, 0.5), B′ (1, 1), C′ (1, 2), D′ (0,1.5)]

The Viewing Pipeline

- **Window:** Area selected in world-coordinate for display is called window. It defines what is to be viewed.
- **Viewport:** Area on a display device in which window image is display (mapped) is called viewport. It defines where to display.
- In many case window and viewport are rectangle, also other shape may be used as window and viewport.
- In general finding device coordinates of viewport from word coordinates of window is called as **viewing transformation.**
- Sometimes we consider this viewing transformation as window-to-viewport transformation but in general it involves more steps.

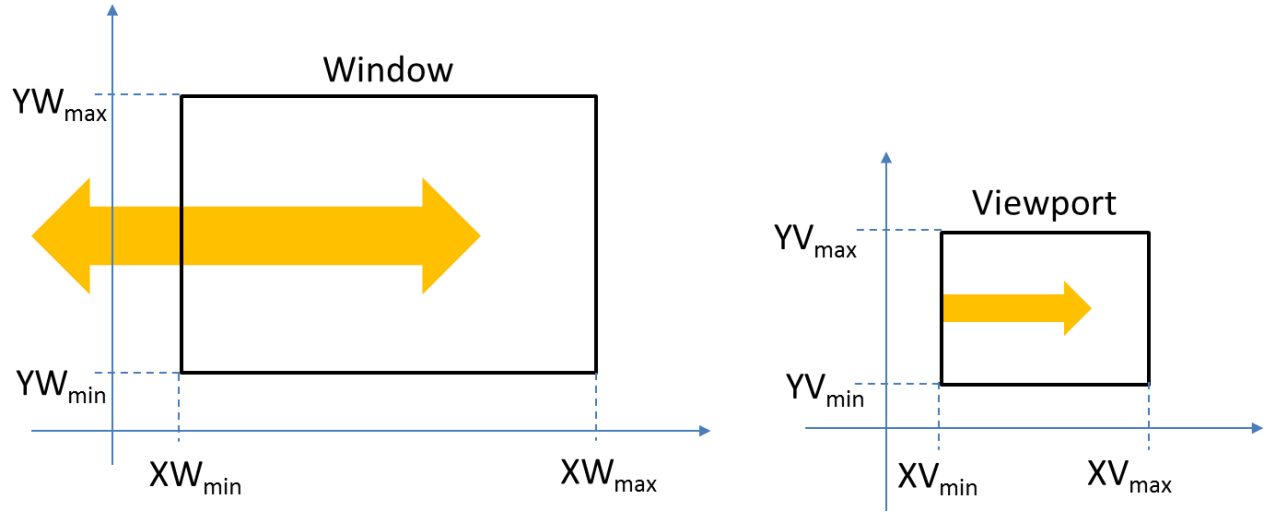


Fig. 3.1: - A viewing transformation using standard rectangles for the window and viewport.

- Now we see steps involved in viewing pipeline.

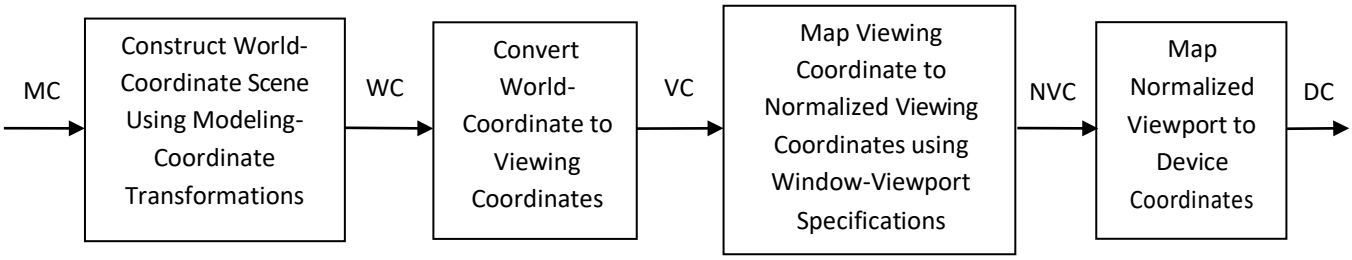


Fig. 3.2: - 2D viewing pipeline.

- As shown in figure above first of all we construct world coordinate scene using modeling coordinate transformation.
- After this we convert viewing coordinates from world coordinates using window to viewport transformation.
- Then we map viewing coordinate to normalized viewing coordinate in which we obtain values in between 0 to 1.
- At last we convert normalized viewing coordinate to device coordinate using device driver software which provide device specification.
- Finally device coordinate is used to display image on display screen.
- By changing the viewport position on screen we can see image at different place on the screen.
- By changing the size of the window and viewport we can obtain zoom in and zoom out effect as per requirement.
- Fixed size viewport and small size window gives zoom in effect, and fixed size viewport and larger window gives zoom out effect.
- View ports are generally defines with the unit square so that graphics package are more device independent which we call as normalized viewing coordinate.

Viewing Coordinate Reference Frame

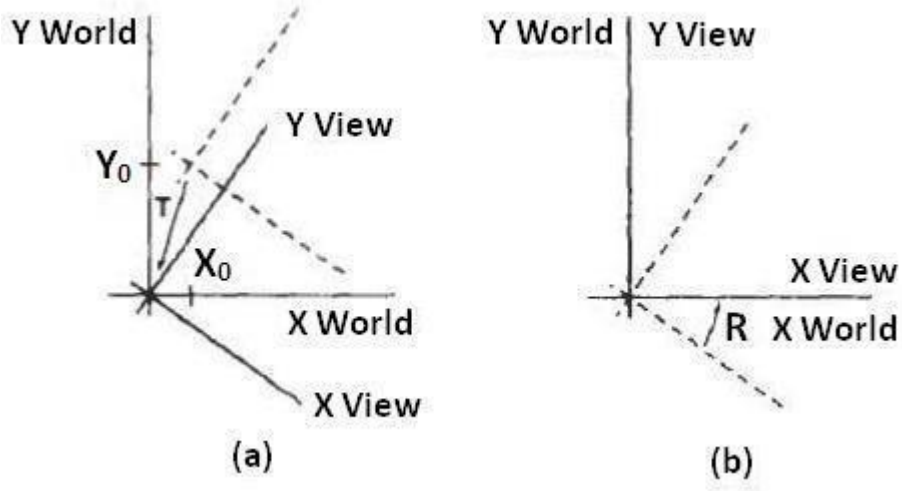


Fig. 3.3: - A viewing-coordinate frame is moved into coincidence with the world frame in two steps: (a) translate the viewing origin to the world origin, and then (b) rotate to align the axes of the two systems.

- We can obtain reference frame in any direction and at any position.
- For handling such condition first of all we translate reference frame origin to standard reference frame origin and then we rotate it to align it to standard axis.
- In this way we can adjust window in any reference frame.
- this is illustrate by following transformation matrix:

$$\mathbf{M_{wc,vc} = RT}$$

- Where T is translation matrix and R is rotation matrix.

Window-To-Viewport Coordinate Transformation

- Mapping of window coordinate to viewport is called window to viewport transformation.
- We do this using transformation that maintains relative position of window coordinate into viewport.
- That means center coordinates in window must be remains at center position in viewport.
- We find relative position by equation as follow:

$$\frac{\mathbf{x_v - x_{vmin}}}{\mathbf{x_{vmax} - x_{vmin}}} = \frac{\mathbf{x_w - x_{wmin}}}{\mathbf{x_{wmax} - x_{wmin}}}$$

$$\frac{\mathbf{y_v - y_{vmin}}}{\mathbf{y_{vmax} - y_{vmin}}} = \frac{\mathbf{y_w - y_{wmin}}}{\mathbf{y_{wmax} - y_{wmin}}}$$

- Solving by making viewport position as subject we obtain:

$$\mathbf{x_v = x_{vmin} + (x_w - x_{wmin})s_x}$$
$$\mathbf{y_v = y_{vmin} + (y_w - y_{wmin})s_y}$$

- Where scaling factor are:

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

- We can also map window to viewport with the set of transformation, which include following sequence of transformations:
  1. Perform a scaling transformation using a fixed-point position of (x_{Wmin},y_{wmin}) that scales the window area to the size of the viewport.
  2. Translate the scaled window area to the position of the viewport.
- For maintaining relative proportions we take ($s_x = s_y$). in case if both are not equal then we get stretched or contracted in either the x or y direction when displayed on the output device.
- Characters are handle in two different way one way is simply maintain relative position like other primitive and other is to maintain standard character size even though viewport size is enlarged or reduce.
- Number of display device can be used in application and for each we can use different window-to-viewport transformation. This mapping is called the **workstation transformation.**

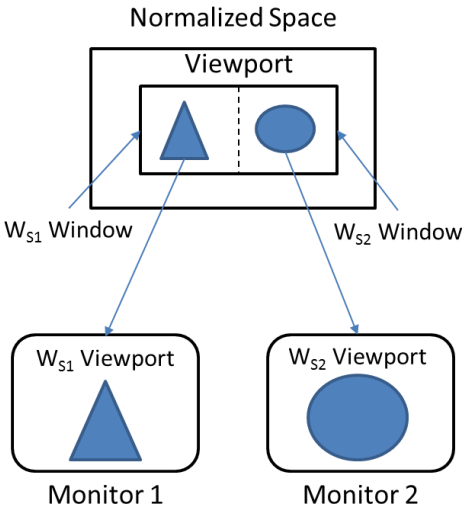


Fig. 3.4: - workstation transformation.

- As shown in figure two different displays devices are used and we map different window-to-viewport on each one.

Clipping Operations

- Generally, 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 simply **clipping**. The region against which an object is to clip is called a **clip window.**
- Clip window can be general polygon or it can be curved boundary.

Application of Clipping

- It can be used for displaying particular part of the picture on display screen.
- Identifying visible surface in 3D views.
- Antialiasing.
- Creating objects using solid-modeling procedures.
- Displaying multiple windows on same screen.
- Drawing and painting.

Point Clipping

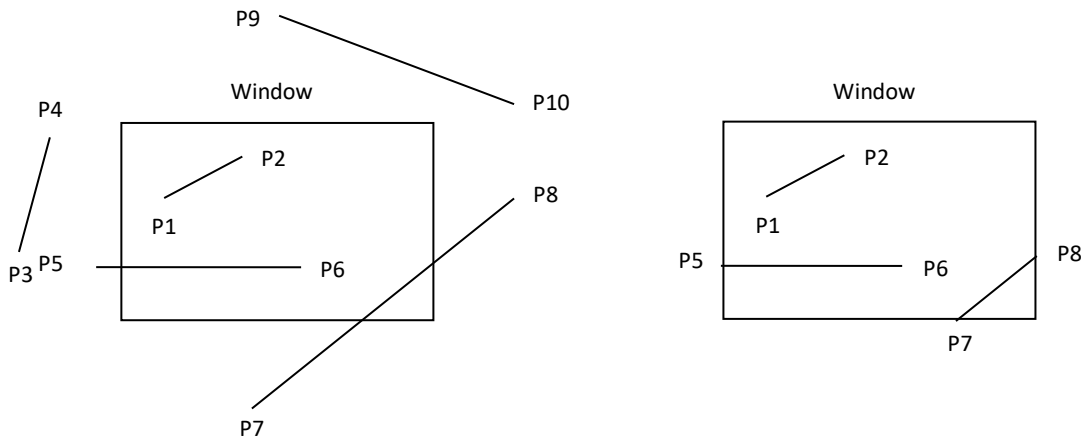
- In point clipping we eliminate those points which are outside the clipping window and draw points which are inside the clipping window.
- Here we consider clipping window is rectangular boundary with edge (x_{wmin},x_{wmax},y_{wmin},y_{wmax}).
- So for finding wether given point is inside or outside the clipping window we use following inequality:

  $\boldsymbol{x_{wmin} \leq x \leq x_{wamx}}$

  $\boldsymbol{y_{wmin} \leq y \leq y_{wamx}}$
- If above both inequality is satisfied then the point is inside otherwise the point is outside the clipping window.

Line Clipping

- Line clipping involves several possible cases.
  1. Completely inside the clipping window.
  2. Completely outside the clipping window.
  3. Partially inside and partially outside the clipping window.



Before Clipping (a)

After Clipping (b)

Fig. 3.5: - Line clipping against a rectangular window.

- Line which is completely inside is display completely. Line which is completely outside is eliminated from display. And for partially inside line we need to calculate intersection with window boundary and find which part is inside the clipping boundary and which part is eliminated.
- For line clipping several scientists tried different methods to solve this clipping procedure. Some of them are discuss below.

Cohen-Sutherland Line Clipping

- This is one of the oldest and most popular line-clipping procedures.

Region and Region Code

- In this we divide whole space into nine region and assign 4 bit code to each endpoint of line depending on the position where the line endpoint is located.

1001	1000	1010
0001	0000	0010
0101	0100	0110

Fig. 3.6: - Workstation transformation.

- Figure 3.6 shows code for line end point which is fall within particular area.
- Code is deriving by setting particular bit according to position of area.
  Set bit 1: For left side of clipping window.
  Set bit 2: For right side of clipping window.
  Set bit 3: For below clipping window.
  Set bit 4: For above clipping window.
- All bits as mention above are set means 1 and other are 0.

Algorithm

Step-1:

Assign region code to both endpoint of a line depending on the position where the line endpoint is located.

Step-2:

If both endpoint have code '0000'

Then line is completely inside.

Otherwise

Perform logical ending between this two codes.

If result of logical ending is non-zero

Line is completely outside the clipping window.

Otherwise

Calculate the intersection point with the boundary one by one.

Divide the line into two parts from intersection point.

Recursively call algorithm for both line segments.

Step-3:
Draw line segment which are completely inside and eliminate other line segment which found completely outside.

Intersection points calculation with clipping window boundary

- For intersection calculation we use line equation "$y = mx + b$".
- 'x' is constant for left and right boundary which is:
  - for left "$x = x_{wmin}$"
  - for right "$x = x_{wmax}$"
- So we calculate y coordinate of intersection for this boundary by putting values of x depending on boundary is left or right in below equation.
  $\boldsymbol{y = y_1 + m(x - x_1)}$
- 'y' coordinate is constant for top and bottom boundary which is:
  - for top "$y = y_{wmax}$"
  - for bottom "$y = y_{wmin}$"
- So we calculate x coordinate of intersection for this boundary by putting values of y depending on boundary is top or bottom in below equation.
  $\boldsymbol{x = x_1 + \frac{y - y_1}{m}}$

Liang-Barsky Line Clipping

- Line clipping approach is given by the Liang and Barsky is faster than cohen-sutherland line clipping. Which is based on analysis of the parametric equation of the line which are as below.
  $\boldsymbol{x = x_1 + u\Delta x}$
  $\boldsymbol{y = y_1 + u\Delta y}$
  Where $0 \leq u \leq 1$,$\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$.

Algorithm

1. Read two end points of line $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$
2. Read two corner vertices, left top and right bottom of window: (x_{wmin}, y_{wmax}) and (x_{wmax}, y_{wmin})
3. Calculate values of parameters p_k and q_k for $k = 1, 2, 3, 4$ such that,
   $p_1 = -\Delta x, \quad q_1 = x_1 - x_{wmin}$
   $p_2 = \Delta x, \quad q_2 = x_{wmax} - x_1$
   $p_3 = -\Delta y, \quad q_3 = y_1 - y_{wmin}$
   $p_4 = \Delta y, \quad q_4 = y_{wmax} - y_1$
4. If $p_k = 0$ for any value of $k = 1, 2, 3, 4$ then,
   Line is parallel to k^{th} boundary.

   If corresponding $q_k < 0$ then,
   Line is completely outside the boundary. Therefore, discard line segment and Go to Step 10.
   Otherwise
   Check line is horizontal or vertical and accordingly check line end points with corresponding boundaries.

   If line endpoints lie within the bounded area
   Then use them to draw line.
   Otherwise

Use boundary coordinates to draw line. And go to Step 8.

5. For $k = 1,2,3,4$ calculate r_k for nonzero values of p_k and q_k as follows:
   $r_k = \frac{q_k}{p_k}, for\ k = 1,2,3,4$
6. Find $u_1 and\ u_2$ as given below:
   $u_1 = \max\{0, r_k | where\ k\ takes\ all\ values\ for\ which\ p_k < 0\}$
   $u_2 = \min\{1, r_k | where\ k\ takes\ all\ values\ for\ which\ p_k > 0\}$
7. If $u_1 \leq u_2$ then
   Calculate endpoints of clipped line:
   $x_1' = x_1 + u_1 \Delta x$
   $y_1' = y_1 + u_1 \Delta y$
   $x_2' = x_1 + u_2 \Delta x$
   $y_2' = y_1 + u_2 \Delta y$
   Draw line (x_1', y_1', x_2', y_2');
8. Stop.

Advantages

1. More efficient.
2. Only requires one division to update u_1 and u_2.
3. Window intersections of line are calculated just once.

Nicholl-Lee-Nicholl Line Clipping

- By creating more regions around the clip window the NLN algorithm avoids multiple clipping of an individual line segment.
- In Cohen-Sutherlan line clipping sometimes multiple calculation of intersection point of a line is done before actual window boundary intersection or line is completely rejected.
- These multiple intersection calculation is avoided in NLN line clipping procedure.
- NLN line clipping perform the fewer comparisons and divisions so it is more efficient.
- But NLN line clipping cannot be extended for three dimensions while Cohen-Sutherland and Liang-Barsky algorithm can be easily extended for three dimensions.
- For given line we find first point falls in which region out of nine region shown in figure below but three region shown in figure by putting point are only considered and if point falls in other region than we transfer that point in one of the three region.

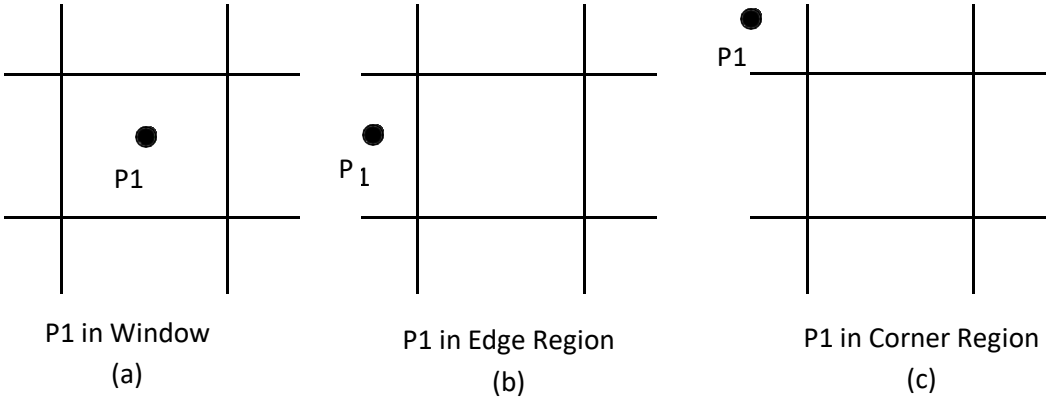


Fig. 3.7: - Three possible position for a line endpoint p1 in the NLN line-clipping algorithm.

- We can also extend this procedure for all nine regions.
- Now for p1 is inside the window we divide whole area in following region:

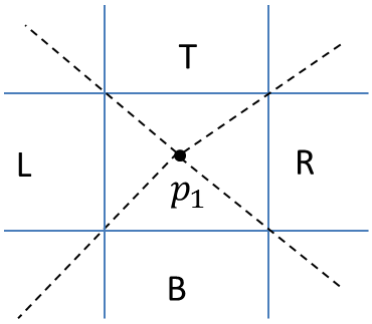


Fig. 3.8: - Clipping region when p1 is inside the window.

- Now for p1 is in edge region we divide whole area in following region:

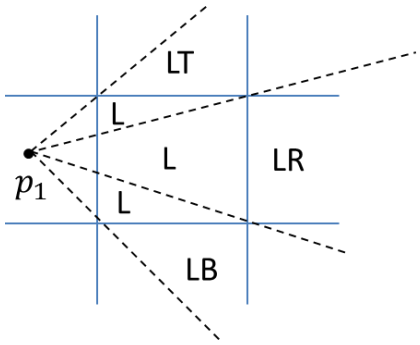


Fig. 3.9: - Clipping region when p1 is in edge region.

- Now for p1 is in corner region we divide whole area in following region:

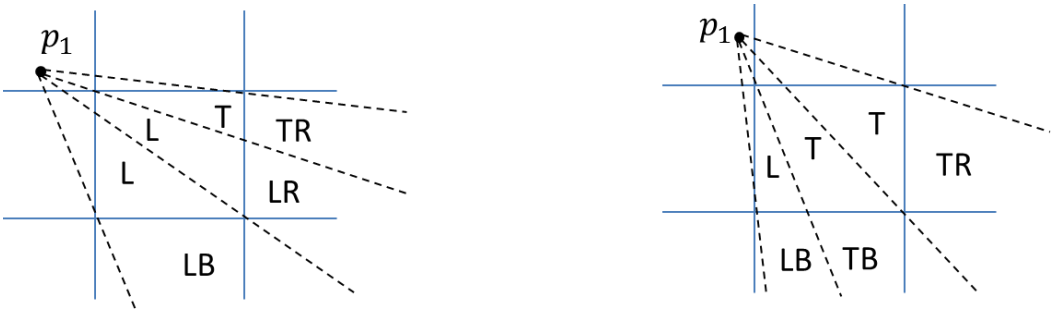


Fig. 3.10: - Two possible sets of clipping region when p1 is in corner region.

- Regions are name in such a way that name in which region p2 falls is gives the window edge which intersects the line.
- For example region LT says that line need to clip at left and top boundary.
- For finding that in which region line $\boldsymbol{p_1p_2}$ falls we compare the slope of the line to the slope of the boundaries:

  $\boldsymbol{slope\ \overline{p_1p_{B1}} < slope\ \overline{p_1p_2} < slope\ \overline{p_1p_{B2}}}$

  Where $\overline{\boldsymbol{p_1p_{B1}}}$ and $\overline{\boldsymbol{p_1p_{B2}}}$ are boundary lines.
- For example p1 is in edge region and for checking whether p2 is in region LT we use following equation.

  $\boldsymbol{slope\ \overline{p_1p_{TR}} < slope\ \overline{p_1p_2} < slope\ \overline{p_1p_{TL}}}$

  $$\frac{\boldsymbol{y_T - y_1}}{\boldsymbol{x_R - x_1}} < \frac{\boldsymbol{y_2 - y_1}}{\boldsymbol{x_2 - x_1}} < \frac{\boldsymbol{y_T - y_1}}{\boldsymbol{x_L - x_1}}$$
- After checking slope condition we need to check weather it crossing zero, one or two edges.
- This can be done by comparing coordinates of p_2 with coordinates of window boundary.
- For left and right boundary we compare x coordinates and for top and bottom boundary we compare y coordinates.
- If line is not fall in any defined region than clip entire line.

- Otherwise calculate intersection.
- After finding region we calculate intersection point using parametric equation which are:
- $\boldsymbol{x = x_1 + (x_2 - x_1)u}$
- $\boldsymbol{y = y_1 + (y_2 - y_1)u}$
- For left or right boundary $x = x_l or\ x_r$ respectively, with $u = (x_{l/r} - x1)/\ (x2 - x1)$, so that y can be obtain from parametric equation as below:
- $\boldsymbol{y = y_1 + \frac{y^2 - y_1}{x_2 x_1}(x_L - x_1)}$
- Keep the portion which is inside and clip the rest.

Polygon Clipping

- For polygon clipping we need to modify the line clipping procedure because in line clipping we need to consider about only line segment while in polygon clipping we need to consider the area and the new boundary of the polygon after clipping.

Sutherland-Hodgeman Polygon Clipping

- For correctly clip a polygon we process the polygon boundary as a whole against each window edge.
- This is done by whole polygon vertices against each clip rectangle boundary one by one.
- Beginning with the initial set of polygon vertices we first clip against the left boundary and produce new sequence of vertices.
- Then that new set of vertices is clipped against the right boundary clipper, a bottom boundary clipper and a top boundary clipper, as shown in figure below.

Fig. 3.11: - Clipping a polygon against successive window boundaries.

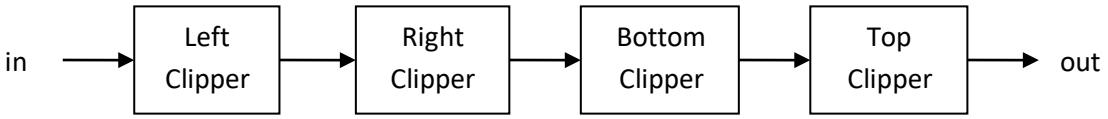


Fig. 3.12: - Processing the vertices of the polygon through boundary clipper.

- There are four possible cases when processing vertices in sequence around the perimeter of a polygon.

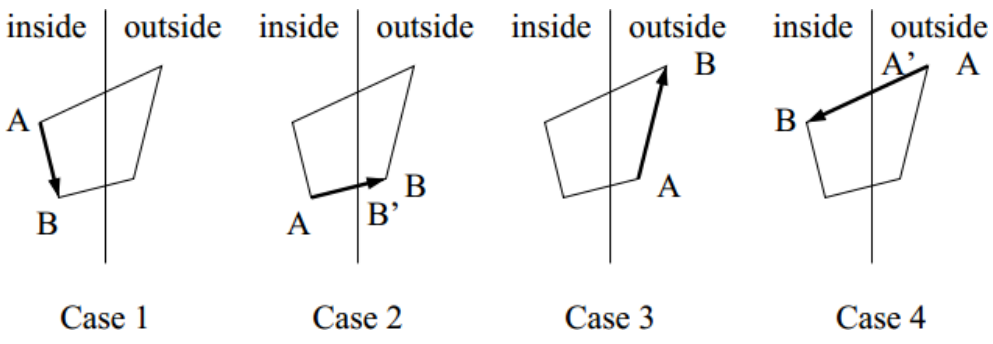


Fig. 3.13: - Clipping a polygon against successive window boundaries.

- As shown in case 1: if both vertices are inside the window we add only second vertices to output list.
- In case 2: if first vertices is inside the boundary and second vertices is outside the boundary only the edge intersection with the window boundary is added to the output vertex list.
- In case 3: if both vertices are outside the window boundary nothing is added to window boundary.
- In case 4: first vertex is outside and second vertex is inside the boundary, then adds both intersection point with window boundary, and second vertex to the output list.
- When polygon clipping is done against one boundary then we clip against next window boundary.
- We illustrate this method by simple example.

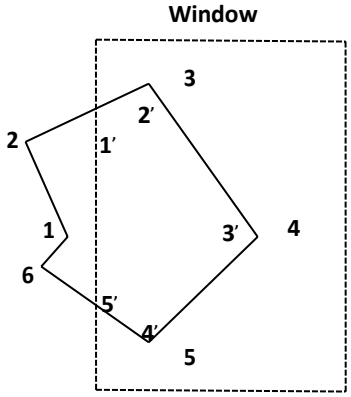


Fig. 3.14: - Clipping a polygon against left window boundaries.

- As shown in figure above we clip against left boundary vertices 1 and 2 are found to be on the outside of the boundary. Then we move to vertex 3, which is inside, we calculate the intersection and add both intersection point and vertex 3 to output list.
- Then we move to vertex 4 in which vertex 3 and 4 both are inside so we add vertex 4 to output list, similarly from 4 to 5 we add 5 to output list, then from 5 to 6 we move inside to outside so we add intersection pint to output list and finally 6 to 1 both vertex are outside the window so we does not add anything.
- Convex polygons are correctly clipped by the Sutherland-Hodgeman algorithm but concave polygons may be displayed with extraneous lines.
- For overcome this problem we have one possible solution is to divide polygon into numbers of small convex polygon and then process one by one.
- Another approach is to use Weiler-Atherton algorithm.

Weiler-Atherton Polygon Clipping

- In this algorithm vertex processing procedure for window boundary is modified so that concave polygon also clip correctly.
- This can be applied for arbitrary polygon clipping regions as it is developed for visible surface identification.
- Main idea of this algorithm is instead of always proceeding around the polygon edges as vertices are processed we sometimes need to follow the window boundaries.
- Other procedure is similar to Sutherland-Hodgeman algorithm.
- For clockwise processing of polygon vertices we use the following rules:
  - For an outside to inside pair of vertices, follow the polygon boundary.
  - For an inside to outside pair of vertices, follow the window boundary in a clockwise direction.
- We illustrate it with example:

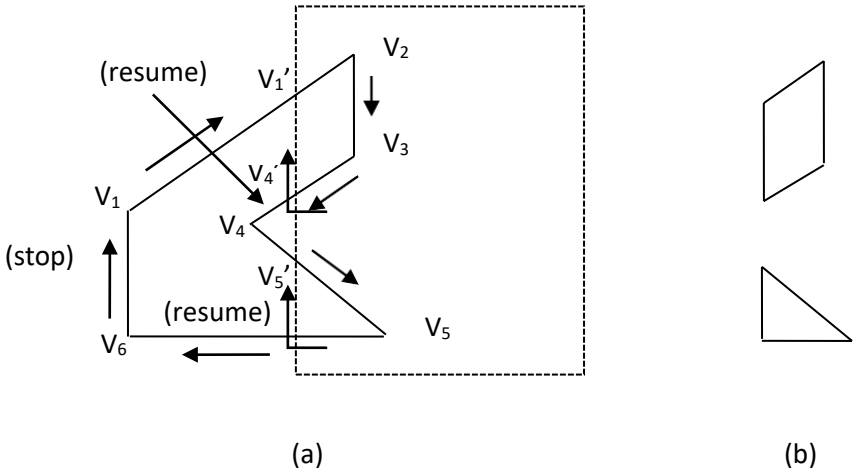


Fig. 3.14: - Clipping a concave polygon (a) with the Weiler-Atherton algorithm generates the two se

- As shown in figure we start from v1 and move clockwise towards v2 and add intersection point and next point to output list by following polygon boundary, then from v2 to v3 we add v3 to outputlist.
- From v3 to v4 we calculate intersection point and add to output list and follow window boundary.
- Similarly from v4 to v5 we add intersection point and next point and follow the polygon boundary, next we move v5 to v6 and add intersection point and follow the window boundary, and finally v6 to v1 is outside so no need to add anything.
- This way we get two separate polygon section after clipping.