2.2. CATHODE RAY TUBE (CRT)

The simplest version of a cathode ray tube consists of a gas-filled glass tube in which two metal plates, one negatively charged (the cathode) and the other positively charged (the anode), have been placed. When a very large voltage is placed across the electrodes, the neutral gas inside the tube will ionize into conducting plasma, and a current will flow as electrons travel from the cathode to the other side.

A cathode ray tube (CRT) is a type of analog display device. Cathode ray tubes are special, electronic vacuum tubes that use focused electron beams to display images. Though tubes of this type are used for many purposes. Cathode ray tubes are most famous for their use in such things as televisions, oscilloscopes, computer and radar displays, and automated teller machines. Cathode ray tubes are also used in video game equipment.

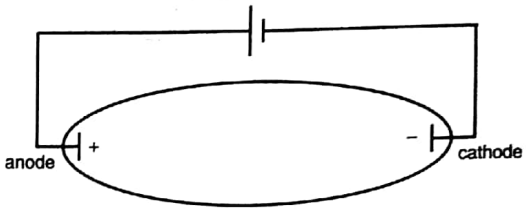


Fig. 2.1.

A cathode ray tube has a cathode or negatively charged terminal. In a cathode ray tube, this terminal is a heated filament, much like the filament seen in a light bulb. The filament is contained inside a vacuum with a glass tube. Inside the tube, a beam of electrons is allowed to flow from the filament into the vacuum. The flow of the electrons is natural, not forced.

When used inside a television set, a CRT's electrons are concentrated in a light beam by a positively charged terminal, called an anode. An accelerating anode is then used to speed up the movement of the electrons. These fast-moving electrons fly through the tube's vacuum, hitting the phosphor-coated screen and making it glow.

Cathode-ray tubes are found in oscilloscopes, and similar devices are used in TV picture tubes and computer displays. The name goes back to the early 1900s. Cathode-ray tubes use an electron beam; before the basic nature of the beam was understood, it was called a cathode ray because it originated from the cathode (negative electrode) of a vacuum.

Figure 2.2 is a schematic diagram of the principal elements of a cathode-ray tube. The interior of the tube is a very good vacuum, with a pressure of around 0.01 Pa (10-7 atm) or less. At any greater pressure, collisions of electrons with air molecules would scatter the electron beam excessively.

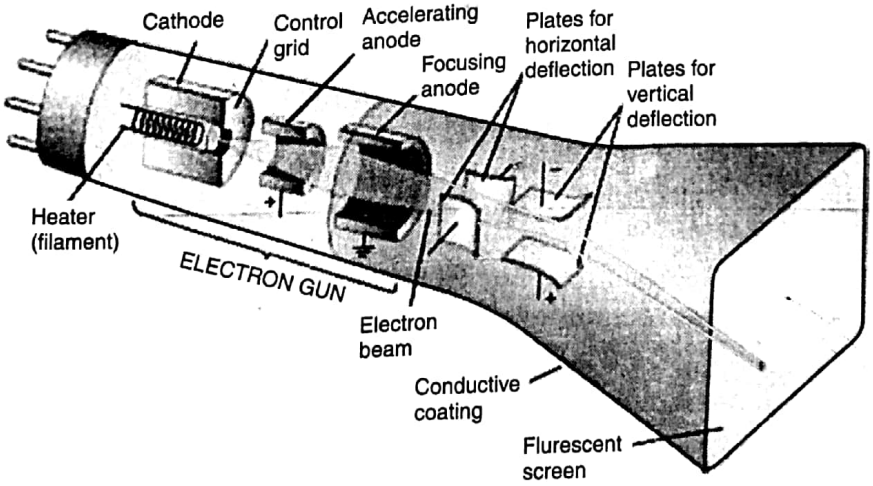


Fig. 2.2.

The cathode, at the left end in the figure, is raised to a high temperature by the heater, and electrons evaporate from the surface of the cathode. The accelerating anode, with a small

Scanned by CamScanner

hole at its center, is maintained at a high potential V_1, of the order of 1 to 20 kV, relative to the cathode. This potential difference gives rise to an electric field directed from right to left in the region between the accelerating anode and the cathode. Electrons passing through the hole in the anode form an arrow beam and travel with constant horizontal velocity from the anode to the fluorescent screen. The area where the electrons strike the screen glows brightly. The control grid regulates the number of electrons that reach the anode and hence the brightness of the spot on the screen. The focusing anode ensures that electrons leaving the cathode in slightly different directions are focused down to a narrow beam and all arrive at the same spot on the screen. The assembly of cathode, control grid, focusing anode, and accelerating electrode is called the *electron gun*.

The beam of electrons passes between two pairs of deflecting plates. An electric field between the first pair of plates deflects the electrons horizontally, and an electric field between the second pair deflects them vertically. If no deflecting fields are present, the electrons travel in a straight line from the hole in the accelerating anode to the center of the screen, where they produce a bright spot.

The electron gun fixed at the base plate of the CRT focuses electron beam on the CRT plate which is coated with phosphor bronze. The electron beam from the gun sequentially scans every pixel of the raster. In the process the beam hits a pixel and the phosphor dot of the pixel gets excited. As the electron beam moves to the immediate next dot, the currently excited dot starts emitting light energy in order to come back to its original static state. As the light emitted by the phosphor fades very rapidly, some method is needed for maintaining the screen picture. One way to *keep* the phosphor glowing is to redrawn the picture repeatedly by quickly directing the electron beam back over the same points. This type of display is called a **refresh CRT**.

> **Note:** The phosphors used in a graphic display are normally chosen for their colour characteristics and persistence. Ideally the persistence, measured as the time for the brightness to drop to 1/10 of its initial value, should last about 100 milliseconds or less, allowing refresh at 30 hertz rates without noticeable smearing as the image moves. The phosphor should also possess a number of other attributes such as small grain size for added resolution, high efficiency in terms of electric energy converted to light, and resistance to burning under prolonged excitation. To improve performance in one or another of these respects, many different phosphors have been produced. There phosphors are identified by a numbering system using names like P1, P4, P7 etc.
>
> The most popular phosphors for graphic displays are P7, a fairly long persistence blue phosphor that leaves a green after glow, and P31, which is green and has a much shorter resistance black and white television tubes generally use P4, a white phosphor with about same resistance as P31.

(a) The Beam- Penetration CRT

The normal CRT can generate images of only a single colour due to limitations of its phosphor. A colour CRT device uses a multilayer phosphor and achieves colour control by modulating a normally constant parameter, namely the *beam accelerating potential*. The screen is coated with a layer of green phosphor over which a layer of red phosphor is desposited. When a low potential electron beam strikes the screen only the red phosphor is excited thus producing a red trace. A higher velocity beam will penetrate into the green-phosphor increasing the green component of the light output.

By varying the beam potential different combinations of red and green light can produce a limited range of colour such as orange, yellow etc. The speed of the electrons, and hence the screen colour at any point, is controlled by the beam-acceleration voltage. Beam penetration

has been an inexpensive way to produce colour in random-scan monitors but only four colours are possible and quality of picture is not as good as with other methods.

Advantages: The biggest advantage is that it is at half cost of shadow mask and its resolution is better.

Disadvantage: The main problem arises at the time of switching colours, when the beam-accelerating potential needs to be changed by significant amounts in order to prevent smear and flicker of the image. The hardware or the software must be designed to introduce adequate delays between colour changes, so that there is time for voltage to settle. So biggest disadvantage is that change of colour takes time which doesn't suit interactive graphics at all.

To prevent frequent delays and consequent flicker all the red elements of the picture should be displayed first, then the accelerating potential can be changed to display the yellow elements and so on through all the different colours.

Shadow Mask CRT

Shadow-mask methods are commonly used in raster-scan systems (including colour TV) because they produce a much wider range of colour than the beam penetration method. A shadow-mask CRT has three phosphor colour dots at each pixel position. One phosphor dot emits a red light, another emits a green light, and the third emits a blue light. This type of CRT has three electron guns, one for each colour dot, and a shadow-mask grid just behind the phosphor coated screen. Before the stream of electrons produced by the CRT's cathode reaches the phosphor-coated faceplate, it encounters the shadow mask, a sheet of metal engraved with a pattern of holes. The mask is positioned in the glass funnel of the CRT during manufacture and the phosphor is coated on to the screen so that electrons coming from the red, green and blue gun positions only land on the appropriate phosphor.

Stray electrons strike the shadow mask and are absorbed by it, generating a great deal of heat, which in turn causes the metal to expand. To allow flatter CRTs to be made, the metal most commonly used now for shadow masks is **Invar, an alloy of iron and nickel**. The metal has a low coefficient of expansion and its name derives from the supposed invariability of its dimensions when heat is applied. In reality, its dimensions are not completely invariable and the build up of heat in a shadow mask can let to a form of distortion known as *doming*, where the center of the mask bulges towards the faceplate slightly.

Figure 2.3 illustrates the *delta-delta shadow-mask method*, commonly used in colour CRT systems. The three-electron beam are deflected and focused as a group onto the shadow mask, which contains a series of holes aligned with the

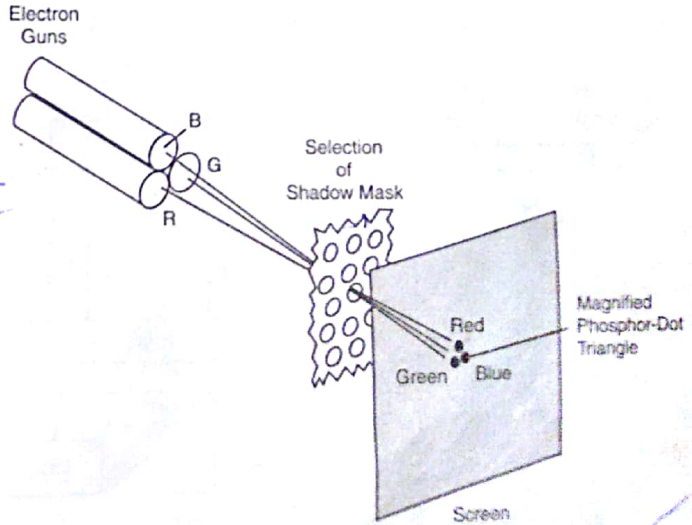


Fig. 2.3.

Scanned by CamScanner

phosphor-dot patterns. When the three beams pass through a hole in the shadow mask, they activate a dot triangle, which appears as a small colour spot on the screen. The phosphor dots in the triangles are arranged so that each electron beam can activate only its corresponding colour dot when it passes through the shadow.

2.3. RANDOM SCAN DISPLAY (MONITORS)

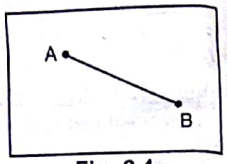


Fig. 2.4.

The original CRT, developed in the late 50's and early 60's, created charts and pictures line by line on the tube surface in any (random) order or direction given, in a vectorial fashion. The electron beam was moved along the particular direction and for the particular length of the line as specified. For this reason the type of device was known as vector calligraphic or stroke. For example if we want a line connecting point A with point B on the vector graphics display, we simply drive the beam deflection circuitry, which will cause beam to go directly from points A to B.

If we want to move the beam from point A to point B without showing a line between points, we can blank the beam as we move it. Thus random scan display generates the image by drawing a set of random straight lines much in the same way one might move a pencil over a piece of paper to draw an image-drawing strokes from one point to another, one line at a time.

There are of course no bit planes containing mapped pixel values in vector system. Instead the **display buffer memory** stores a set of line drawing commands along with end point coordinates in a *display list* or *display program* created by a graphics package. The display processing unit (DPU) executes each command during every refresh cycle and feeds the **vector generator** with digital x, y and Δx, Δy values. The vector generator converts the digital signals into equivalent analog deflection voltages. This causes the electron beam to move to the start point or from the start point to the end point of a line or vector. Thus the beam sweep does not follow any fixed pattern, the direction is arbitrary as dictated by the display commands.

Figure 2.5 shows the typical random scan display architecture. It consists of display controller, central processing unit (CPU), display buffer memory and a CRT.

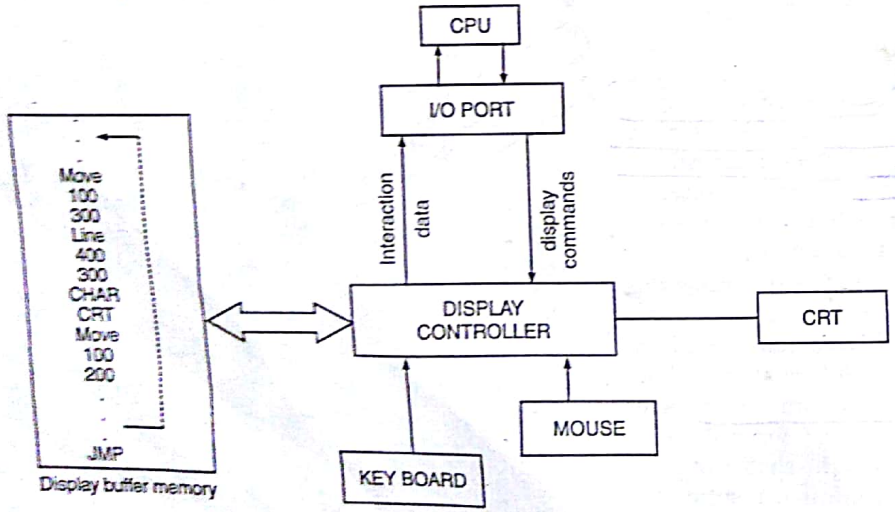


Fig. 2.5. Architecture of random display

A display controller is connected as an I/O peripheral to the CPU. The display buffer memory stores the computer produced display list or display program. The program contains point and line plotting command with end points.

The display controller interprets commands for plotting points, lines and characters and sends digital and point co-ordinates to a vector generator. The vector generator then converts the digital co-ordinate values to analog voltages for beam-deflection circuits that displace an electron beam writing on the CRT's phosphor coating.

Random-scan systems are designed for line drawing applications and cannot display realistic shaded scenes. Devices such as DVST, plasma panel etc. support only line drawing. They do not support for solid areas which can be constructed on raster displays. Since picture definition is stored as a set of line drawing instructions and not as a set of intensity values for all screen points, random displays generally have *higher resolution* than raster systems. Also, vector or random displays produce smooth line drawings because the CRT beam directly follows the line path.

2.4. RASTER SCAN DISPLAYS (MONITORS)

(UPTU 2009-10) (2008-09)

The term RASTER is a synonym for "matrix" therefore a raster scan CRT scans a matrix with electron beam. In a raster-scan system, the electron beam is swept across the screen, one row at a time from top to bottom. As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots. Picture definition is stored in a memory-area called the *refresh buffer* or *frame buffer*. It holds the set of intensity values for all the screen points. The stored intensity values are retrieved from frame buffer and displayed on the screen one row (scan line) at a time. Each screen point is referred to as a pixel. Each pixel on the screen can be specified by its row and column number. Thus by specifying row and column number we can specify the *pixel position* on the screen.

The capability of a raster-scan system to store intensity information for each screen point makes it well suited for the realistic display of scenes containing subtle shading and colour patterns.

Figure 2.6 shows the architecture of a raster display, it consists of display controller, CPU, video controller, refresh buffer, keyboard, mouse and the CRT.

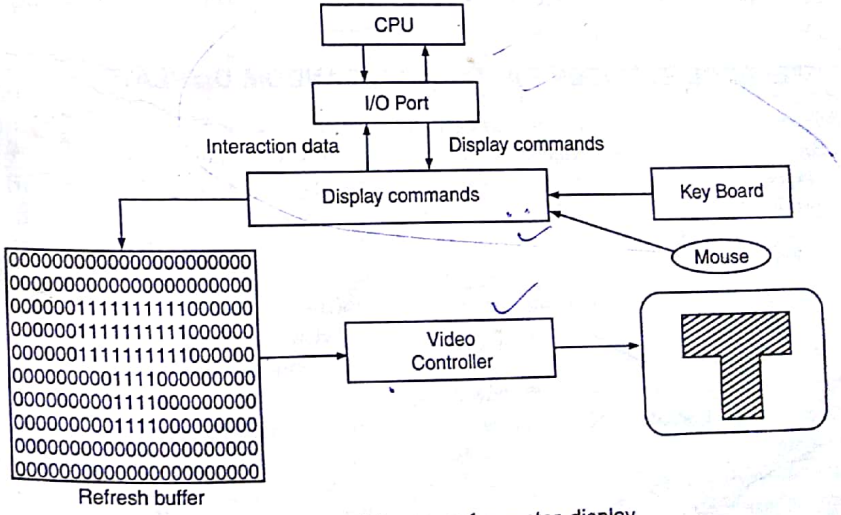


Fig. 2.6. Architecture of a raster display

Scanned by CamScanner

As shown in the figure 2.6, the display image is stored in the from of 1's and 0's in the refresh buffer. The video controller reads this refresh buffer and produces the actual image on the screen.

The raster scan proceeds as follows: Starting from the top left corner of the screen, the electron gun scans horizontally from left to right, one scan line, that is one row at a time, jumping (without tracing the line) to the left end of the next lower row until the bottom right corner is reached. Then it jumps (again without tracing) to the top left corner and starts again, finishing one complete refresh cycle.

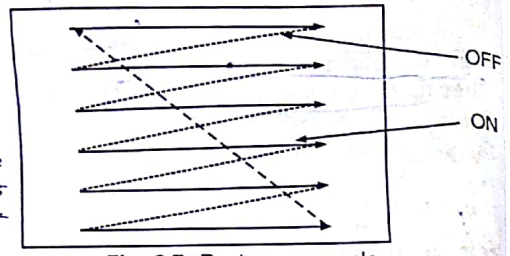


Fig. 2.7. Raster scan cycle

Here, the beam is swept back and forth from the left to the right across the screen. When the beam is moved from the left to the right, it is ON. The beam is OFF, when it is moved from the right to the left as shown by dotted line in figure 2.7.

When the beam reaches the bottom of the screen, it is made OFF and rapidly retraced back to the top left to start again. A display produced in this way is called **raster scan display**. Raster scanning process is similar to reading different lines on the page of a book. After completion of scanning of one line the electron beam files back to the start of next line and process repeats. In the raster scan display, the screen image is maintained by repeatedly scanning the same image. This process is known as *refreshing of screen*. Actually, it is completed in about 1/30th of a second, which is faster than the human eye can perceive thus creating the impression of continuous display and motion. On some raster-scan systems, each frame is displayed in two passes using an **interlaced** refresh procedure. In this, instead of refreshing every line of the screen, when in an interlaced mode the electron guns sweep alternate lines on each pass. In the first pass, odd-numbered lines are refreshed, and in the second pass, even-numbered lines are refreshed. This allows the refresh rate to be doubled because only half the screen is redrawn at a time.

Interlacing is primarily used with slower refreshing rates. On an older, 30 frame per second, noninterlaced display, for instance some flicker is noticeable. But with interlacing each of the two passes can be accomplished in 1/60th of a second, which brings the refresh rate nearer to 60 frames per second.

2.5. DIFFERENCE BETWEEN RASTER AND RANDOM DISPLAYS

The difference between raster and random display is that raster display gives a realistic image. Random displays generally have higher resolution than raster systems. Vector displays produce smooth line drawing because the CRT beam directly follows the line path. A raster system produces *jagged lines* that are plotted as the point sets.

Vector scan display	*Raster Scan Display*
1. Vector display only draws lines and characters.	1. Raster display has ability to display areas filled with solid colours or patterns.
2. Don't use interlacing.	2. Uses interlacing.
3. In vector scan display the beam is moved between the end points of the graphics primitives.	3. In raster scan display the beam is moved all over the screen one scan line at a time from top to bottom and then back to top.
4. Higher resolution.	4. Lower resolution.
5. More expensive.	5. Less expensive.
6 Uses monochrome or beam-penetration type.	6. Uses monochrome or shadow mask type.
7. Vector display draws a continuous and smooth lines.	7. Raster display can display mathematically smooth lines polygons and boundaries of curved primitives only by approximating them with pixel on the raster grid.
8. Editing is easy.	8. Editing is difficult.
9. Refresh rate depends directly on picture complexity.	9. Refresh rate independent of picture complexity.
10. Scan conversion is not required.	10. Graphics primitives are specifies in terms of their end points and must be scan converted into their corresponding pixels in the frame buffer.

2.6. DIRECT-VIEW STORAGE TUBE

(UPTU 2008-09)

Both in the raster scan and random scan system the screen image is maintained (flicker free) by redrawing or refreshing the screen many times per second by cycling through the picture data stored in the refresh buffer. A direct-view storage tube (DVST) gives the alternative method of maintaining the screen image. A DVST uses the storage grid which stores the picture information as a charge distribution just behind the phosphor-coated screen. Figure shows the diagram of a DVST.

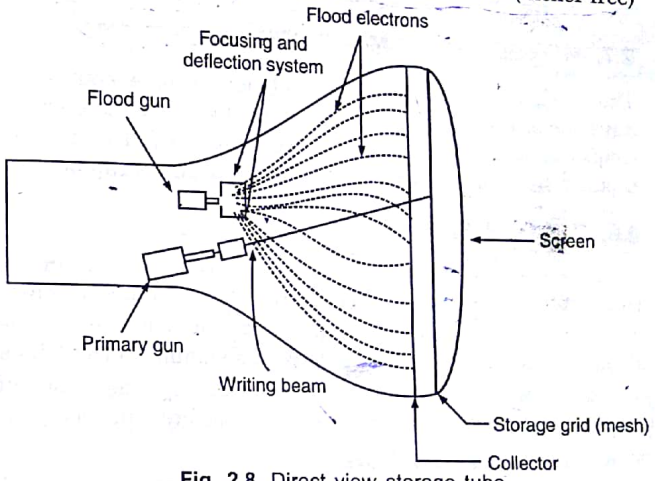


Fig. 2.8. Direct view storage tube

It consists of two electron guns:

(*a*) Primary gun

(*b*) Flood gun

A primary gun stores the picture pattern and flood gun maintains the picture display.

In DVST, there is no refresh buffer, the images are created by drawing vectors or line segments with a relatively slow-moving electron beam. The beam is designed not to draw directly on phosphor but on a fine wire mesh (called storage mesh) coated with dielectric and mounted just behind the screen. A pattern of positive charge is deposited on the grid, and this pattern is transferred to the phosphor coated screen by a continuous flood of electrons emanating from a separate flood gun.

The DVST contains also a second grid just behind the storage mesh, that is called **collector**. The main purpose of *collector* is to smooth out the flow of flood electrons. These electrons pass through the collector at low velocity and are attracted to the positively charged portions of the storage mesh but repelled by the rest. Electrons not repelled by the storage mesh pass right through it and strike the phosphor.

In order to create the bright picture, energy of electrons can be controlled by maintaining the screen at high positive potential by means of voltage applied to thin aluminium coating between the tube face and the phosphor.

Scanned by CamScanner

Advantages of DVST

1. Refreshing of CRT is not required.
2. Because no refreshing is required, very complex pictures can be displayed at very high resolution without flicker.
3. It has flat screen.

Disadvantages of DVST

1. They do not display colours and are available with single level of line intensity.
2. Erasing requires removal of charge on the storage grid. Thus erasing and redrawing process takes several seconds.
3. Selective or part erasing of screen is not possible.
4. Erasing of screen produces unpleasant flash over the entire screen surface which prevents its use of dynamic graphics applications.
5. It has poor contrast as a result of the comparatively low accelerating potential applied to the flood electrons.
6. The performance of DVST is some what inferior to the refresh CRT.

2.7. MONOCHROME MONITORS

The construction and functioning of monochrome monitors have already been described. They have one electron gun. In advanced monochrome monitors, called *gray scale monitors*, various shades of gray ranging from pure white to full black instead of colours, may be used to produce half-tone pictures resulting in a photographic effect.

2.8. COLOUR MONITORS

Colour appeared on the scene after the raster scan technology evolved. Colour monitors have three electron guns, one each for the three primary colours red, green and blue (RGB). The ray strike the tube's inner surface coating consisting of triple clusters of RGB phosphors, through a shadow mask of hole corresponding to the triads. (triple clusters)

In recent years colour monitor technology has improved and costs have come down so much that monochrome monitors are obsolete and colour monitors are standard equipment.

Types of Colour Monitors

(a) Colour Graphics Adapter (CGA): This gave colour capability to PCs, but reduced the 9 × 14 character representation to 8 × 8, resulting in poorer resolution than the monochrome adapters. In the 8 × 8 matrix, one row and one column are left blank for surrounding space and only the 7 × 7 matrix is utilized for the character representation. In the pixel mode, 320 × 200 and 640 × 200 resolutions are available.

(b) Enhanced Graphics Adapter (EGA): This increased the resolution to 640 × 350 and further up to 640 × 800. An EGA with a standard monitor gives 8 × 8 character and 640 × 200 pixels. But an EGA with EGA monitor gives 8 × 14 character and 640 × 350 pixels.

(c) Video Graphics Array (VGA): VGA stands for video graphic array and it uses a special video chip to generate display. It is developed by IBM. VGA systems provide a resolution of 720 × 400 pixels. In graphics mode, the resolution is either 640 × 480 (with 16 colour) or 320 × 200 (with 256 colours). The total palette of colours is 2, 62, 144. So VGA offers more graphics, more colours and highest resolution of any graphics adapter before it. Unlike earlier graphics standards for PCs- MDA, CGA, and EGA- VGA uses analog signal rather than digital signals. Consequently, a monitor designed for one of the older standards will not be able to use VGA.

The VGA BIOS (Basic Input Output System) is the control software present in the system ROM for controlling the VGA circuitry, with the BIOS present. Software can give command and function to BIOS without requiring to manipulate the VGA directly. If we use a monochrome monitor with VGA, colour summing to 64-grey shades is done in ROM BIOS. VGA was officially superseded by IBMs XGA standard, but in reality it was superseded by numerous extensions to VGA made by clone manufacturers that come to be known as **Super-VGA (SVGA).**

Table gives the VGA summary.

VGA Summary

Resolution	*Colours*	*Mode*	*Character*	*Vertical*	*Horizontal*
360 × 400	16	Text	40 × 25	70 Hz	31.5 kHz
720 × 400	16	Text	80 × 25	70	31.5
320 × 200	4	Graph	40 × 25	70	31.5
640 × 200	2	Graph	80 × 25	70	31.5
720 × 400	16	Text	80 × 25	70	31.5
320 × 200	16	Graph	40 × 25	70	31.5
640 × 200	16	Graph	80 × 25	70	31.5
640 × 350	4	Graph	80 × 25	70	31.5
640 × 350	16	Graph	80 × 25	70	31.5
640 × 480	2	Graph	80 × 25	60	31.5
640 × 480	16	Graph	80 × 25	60	31.5
320 × 200	256	Graph	40 × 25	70	31.5

(d) Super VGA: Super video graphic Array (SVGA) refers to any enhancement to existing IBM VGA standard. Unlike VGA-a purely IBM defined standard-Super VGA was defined by the *Video Electronics Standards Association (VESA)*, an open consortium set up to promote interoperability and define standards. The original standard set by VESA was that of 800 × 600 resolution and 16 colours. Today cards are produced with 256 colours and even 1024 × 768 resolution. However, the problem lies in the number of graphics display modes, the VGA standards itself have 17 different ways to display an image on the monitor. The number of colours on higher resolution increase in the newer graphics display standard like the supper VGA, IBM's 8514/A adapter and the extended graphics standard (XGA). The new generation video adapters using the continuous edge graphics (CEG) chips can even deliver 7,40,000 colours simultaneously on the monitor.

While the output of a VGA or SVGA video card is analog, the internal calculations the card performs in order to arrive at these output voltages are entirely digital. To increase the number of colours a super VGA display system can reproduce, no change at all is needed for the monitor, but the video card needs to handle much larger numbers and may well need to be redesigned from scratch on paper, the original super VGA was to be succeeded by super XGA, but in practice the industry soon abandoned. The attempt to provide a unique name for each higher display standard, and almost all display systems made between the late 1990's and the early 2000's are classed as super VGA.

2.9. FLAT-PANEL DISPLAY

A number of display methods are in use that is designed to reduce the depth of the CRT display caused by the length of the tube. These devices are collectively known as *flat panel displays*. These types of flat panel displays commonly in use with computer systems are

Scanned by CamScanner

causing the gas (a mix of neon and xenon) in the cells to ionize. This ionized gas (plasma) emits high frequency UV rays, which stimulate the cells phosphors, causing them to glow the desired colour.

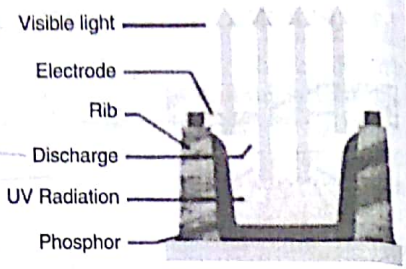


Fig. 2.12.

Each individual plasma cell is switched on and off by its own electrode. A high-definition plasma TV can have up to 6 million cells.

Because a plasma panel is illuminated at the sub-pixel level, images are extremely accurate, and the panel's light output is both high and consistent across the entire screen area. Plasma TVs also provide very wide horizontal and vertical viewing angles. Picture quality looks sharp and bright from virtually anywhere in the room. Because plasma TV screens do use a phosphor coating (like (RT-based TVs), the potential for screen **burn-in** exists, so it is important to follow the manufacturer's recommendation on set up and use.

Advantages and Disadvantages of LCD and Plasma Display

LCD display advantages	*LCD display disadvantages*
• Good colour reproduction • Very thin • Light weight • Perfect sharpness at native resolution • Excellent longevity • No screen burn-in effect	• Fixed resolution • Notorious "screen door" effect on lesser models • Poor contrast ratios (even excellent units have only 700:1) • Very difficult to produce deep blacks • Weak and "stuck" pixels are common • Viewing angle on older models may be narrow • Potential for slower refresh rates than plasma (some newer models are getting better)

Plasma advantages	*Plasma disadvantages*
• Newer models have much better contrast ratios than many direct view TVs • Excellent colour reproduction • Excellent life expectancy • Excellent viewing angle	• Although thin, plasma TVs are fairly heavy (professional installation recommended for on-wall use) • Very susceptible to screen burn-in • Cannot produce deep black levels accurately • Fragile • Use a lot of power

LCD vs Plasma Display Technology

Comparison	Plasma Display	LCD Display	Advantage
Screen size	Screen sizes range from 32 inches to 60 inches	Sizes range from 13 inches to 40 inches, but larger screens are expected soon.	Plasma. Larger LCDs are already in development
Viewing angle	Up to 160°	Up to 170°	LCDs.

Screen Refresh Rates	Plasma displays refresh and handle rapid movements in video about as well as CRT televisions.	LCD TVs were originally designed for data display, and not video. Therefore refresh rates had to be improved. LCD TVs with refresh rates of 16 ms or higher show very little noticeable artifacts.	Slight edge to plasma technology.
Burn-in	Plasma displays can suffer from burn in produced by static images. After extended periods, stationary images "burn in" and produce an after-images ghost which remains permanently on the screen. Newer plasma models have addressed burn-in and reduced the issues of older models.	LCD panels do not suffer from burn-in.	LCDs
Product Life span	Typical plasma displays have a life span of 20,000 to 30,000 hours, which equates to at least two years, three months of 24/7 usage before the image fades to half the original brightness.	LCDs life span is typically 50,000-60-000 hours, which equates to at least 5 years of 24/7 use.	LCDs run nearly twice as long as plasma.
Weight	Plasma displays are fairly heavy, and may need additional supports to be mounted onto a wall.	LCDs weight less than comparably sized plasma screens.	LCDs are considerable lighter.
Durability	Plasmas are very fragile making them tricky to ship and install. Unlike the commercials where plasmas are mounted on the celling, plasmas are best installed by a professional, and should be installed on a wall that can bear a good deal of weight.	Much more durable than plasmas. End users can easily mount an LCD display themselves if desired.	LCDs are far less fragile than plasmas.
Shipping	Due to their fragile nature, plasma displays need to be shipped by speciality carriers. Overnight or fast delivery options are not recommended. Special shipping methods and their heavier weight add to higher shipping costs.	Shipping LCDs is not difficult, and is not as expensive as shipping plasma displays.	LCDs are lighter and far less fragile than plasma displays making shipping easier.

Scanned by CamScanner

a small distance. One of the plates is coated with a conducting material, and the other plate is coated with a resistive material. When the outer plate is touched, it is forced into contact with the inner plate. This contact creats a voltage drop across the resistive plate that is converted to the coordinate values of the selected screen position.

In acoustic type, similar to the light rays, sonic beams are generated from the horizontal and vertical edges of the screen. The sonic beam is obstructed or reflected back by putting a finger in the designed location on the screen. From the time of travel of the beams the location of the fingertip is determined.

2.14.7. Digitizer and Graphics Tablet

A digitizer is locator device used for drawing, painting or interactively selecting coordinate positions on an object Graphics tablet is one such digitizer that consists of a flat surface ranging in size from about 6 by 6 inches up to 48 by 72 inches or more, which can detect the position of a movable stylus, a pen-like drawing apparatus. The image generally does not appear on the tablet itself but, rather is displayed on the computer monitor.

Fig. 2.26. Degitizer

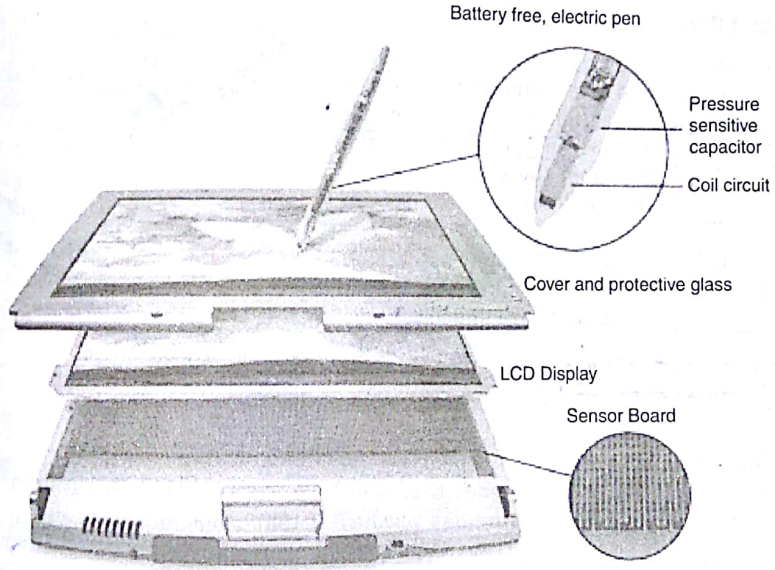


Fig. 2.27. Tablet

The first graphics tablet resembling contemporary tablets was the RAND Tablet, also known as the Grafacon (for Graphic Converter) employed an orthogonal grid of wires under the surface of the pad. When pressure is applied to a point on the tablet using a stylus, the horizontal wire and vertical wire associated with the corresponding grid point meet each other, causing an electric current to flow into each of these wires. Since an electric current is only present in the two wires that meet, a unique coordinate for the stylus can be retrieved. The coordinate returned are tablet coordinates which are converted to the user or screen coordinates by an imaging software. Even if it doesn't touch the tablet, proximity of the stylus to the tablet surface can also be sensed by virtue of a weak magnetic field projected

Scanned by CamScanner

approximately one inch from the tablet surface. It is important to note that, unlike the RAND Tablet, modern tablets do not require electronics in the stylus and any tool that provides an accurate "point" may be used with the pad. In some tablets multiple button hand-cursor is used instead of stylus. Graphics tablets are available in various sizes and price ranges. A6-sized tablets being relatively inexpensive and A3-sized tablets being far more expensive.

Every time user may not wish to enter stylus position into the computer. In such cases user can lift the stylus or make the tablet off by pressing a switch provided on the stylus.

Other graphical tablet technologies use **sound (sonic) coupling and resistive coupling**. The sonic tablet uses sound waves to couple the stylus to microphones positioned on the periphery of the digitizing area. Sound brust are created by an electrical spark at the tip of the stylus. The time between when the spark occurs and when its sound arrives at each microphone is proportional to the distance from the stylus to each microphone. The sonic tablets mainly used in 3D positioning the devices. The resistive tablet uses a battery powered stylus that emits high-frequency radio signals. The tablet is a piece of glass coated with a thin layer of conducting material in which an electrical potential is induced by the radio signals. The strength of the signals at the edges of the tablet is inversely proportional to the distance to the stylus and can thus be used to calculate the stylus position.

2.14.8. Light Pen

A light pen is a pointing device shaped like a pen and is connected to the computer. The tip of the light pen contains a light-sensitive element (photoelectric cell) which, when placed against the screen, detects the light from the screen enabling the computer to identify the location of the pen on the screen.

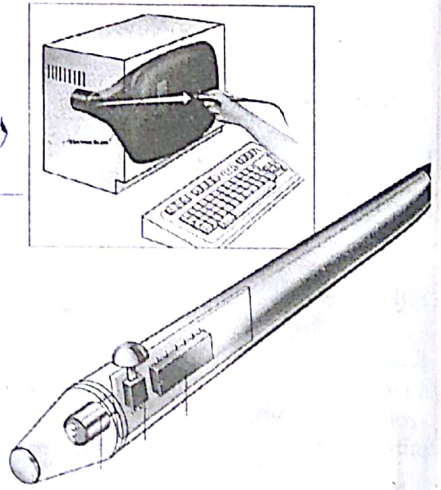


Fig. 2.28. Light pen

Unlike other devices which have associated hardware to track the device and determine x an y values, the light pen needs software support (some kind of tracking program). A light pen can work with any CRT-based monitor but not with LCD screens, projectors or other display devices.

The light pen actually works by sensing the sudden small change in brightness of a point on the screen when the electron gun refreshes that spot. By noting exactly where the scanning has reached at that moment, the x, y position of the pen can be resolved. The pen position is updated on every refresh of the screen.

Although light pens are still with us, they are not much popular because of several disadvantages. One such disadvantage is that when light pen is pointed at the screen, part of the screen image is obscured by the hand and pen. And prolonged use of the light pen can cause arm fatigue. Another disadvantage is that they cannot detect position with in black areas. To be able to select positions in any screen area with a light pen, we must have some nonzero intensity assigned to each screen pixel. In addition, light pens sometimes give false readings due to background lighting in a room.

2.14.9. Data Glove

The *data glove* is an interface device that uses position tracking sensors and fiber optic strands running down each finger and connected to a compatible computer, the movement

Scanned by CamScanner

2. The display file must be interpreted.
3. An explicit action is required to show the contents of frame buffer on some devices.

EXAMPLE 2.1.

Consider three different raster systems with resolutions of 640 × 480, 1280 × 1024 and 2560 × 2048. What size of frame buffer (in bytes) is needed for each of these systems to store 12 bits per pixel ? How much storage is required for each system if 24 bits per pixel are to be stored ? *(UPTU, MCA 2003)*

Solution: Case 1: For 640 × 480,

Total number of pixel required for 640 × 480 resolution = 640 × 480 pixels

So, size of frame-buffer required = 640 × 480 pixels because 1 pixel can store 12 bits.

Therefore size of frame buffer (in bits) = 640 × 480 × 12 bit

$$= \frac{640 \times 480 \times 12}{8} \text{ bytes (1 byte = 8 bit)}$$

= 460800 bytes = 450 KB (1024 bytes = 1 KB)

For 1280 × 1024: Size of frame buffer (in bits)

$$= 1280 \times 1024 \times 12 \text{ bits} = \frac{1280 \times 1024 \times 12}{8} \text{ bytes}$$

= 1920 KB

For 2560 × 2048:

$$\text{Size of frame-buffer} = \frac{2560 \times 2048 \times 12}{8} \text{ bytes} = 7.5 \text{ MB}$$

Case II: When one pixel can store 24 bits then frame-buffer size will be twice.

For **640 × 480,** Frame-Buffer size

$$= \frac{640 \times 480 \times 24}{8} \text{ byte} = 900 \text{ KB}$$

For 1280 × 1024, Frame-buffer size

= 1920 × 2 KB = 3840 kB = 3.75 MB

For 2560 × 2048, Frame-buffer size

= 7.5 × 2 MB = 15 MB.

EXAMPLE 2.2.

Consider a raster system with a resolution of 1024 × 1024. What is the size of the raster (in bytes) needed to storestire 4 bits per pixel ? How much storage is required if 8 bits per pixel are to be stored ? *(UPTU, B. Tech, 2002)*

Solution: Resolution = 1024 × 1024

Case-1: When 1 pixel = 4 bits

Size of raster = 1024 × 1024 × 4 bits

$$= \frac{1024 \times 1024 \times 4}{8} \text{ bytes} \qquad (\because 1 \text{ bytes} = 8 \text{ bits})$$

$$= 512 \times 1024 = 1 \times 2^{19} \text{ bytes}$$

Case-2: When 1 pixel = 8 bits

Scanned by CamScanner

Size of raster = $1024 \times 1024 \times 8$ bits

$= \frac{1024 \times 1024 \times 8}{8}$ bytes $= 1 \times 2^{20}$ bytes

EXAMPLE 2.3.

Consider two raster systems with resolution of 640 by 480 and 1280 by 1024. How many pixels could be accessed per second in each of these systems by a display controller that refreshes the screen at a rate of 60 frames per second ? What is the access time per pixel in each system ?

Solution: **Case-1.** Resolution = 640×480

N. of pixels in one frame = $640 \times 480 = 307200$

Since controller can access 60 frames in one second

Therefore, total no. of pixels accessed = $60 \times 307200 = 18432000$ per sec.

Access time/pixel = 1/total pixels accessed per sec.

$= 5.4 \times 10^{-8}$ sec/pixel.

Case-2. When resolution = 1280×1024

Total no. of pixels accessed by the raster system

$= 60 \times 1280 \times 1024 = 78643200$ per sec.

Access time per pixel = $1/78643200 = 1.2 \times 10^{-8}$ sec/pixel

EXAMPLE 2.4.

A laser printer is capable of printing two pages (size 9 × 11 inch) per second at resolution of 600 pixels per inch. How may be bits per second does such device require ?

(UPTU, MCA, 2002)

Solution: Storage required per page (in pixel)

$= 9 \times 11$ inch/sec

$= 9 \times 11 \times 600 \times 600$ pixel sec $(\because 1 \text{ inch} = 600 \text{ pixels})$

Since laser printer is capable of printing two pages per second bits per second required by the printer.

$= 2 \times 9 \times 11 \times 600 \times 600$ pixels per second

$= 7.128 \times 10^7$ pixel/sec $= 7.128 \times 10^7 \, n$ bits/sec

(Assume 1 pixel = n bits)

EXAMPLE 2.5.

How many k bytes does a frame buffer need in a 600 × 400 pixel ?

Solution: Given resolution is 600×400

Suppose 1 pixel can store n bits then the size of frame buffer

= Resolution × bits per pixel

$= (600 \times 400) \times n$ bits $= 240000 \, n$ bits

$= \frac{240000}{1024 \times 8}$ k bytes $(\because 1\text{kb} = 1024 \text{ bits})$

$= 29.30 \, n$ k bytes.

EXAMPLE 2.6.

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

Solution: We know that,

$$\text{Aspect ratio} = \frac{\text{width}}{\text{height}} = \frac{8 \times 100}{10 \times 100} = \frac{4}{5}$$

$\therefore$ Aspect ratio = 4 : 5

EXAMPLE 2.7.

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

(U.P. Tech 2009-10)

Solution: Here, resolution, 1280×1024

That means system contains 1024 scan lines and each scan line contains of 1280 pixels

and refresh rate = 60 frames/sec.

that means 1 frame takes $\frac{1}{60}$ sec

Since, resolution = 1280×1024

$\therefore$ 1 frame consists of 1024 scan lines.

$\therefore$ 1024 scan lines takes to $\frac{1}{60}$ sec

$\therefore$ 1 scan lines take $\frac{1}{60 \times 1024}$ sec $= 0.058$ sec or 58 m sec.

EXAMPLE 2.8.

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

Solution: Here, resolution = 8 inch × 10 inch

First we convert it in pixel, then

now resolution = 8 × 100 by 10 × 100 pixel = 800 × 1000 pixel

1 pixel can store 6 bits

so, frame buffer size required = $800 \times 1000 \times 6$ bits

$= \frac{800 \times 1000 \times 6}{8}$ bytes $= 6 \times 10^5$ bytes.

EXAMPLE 2.9.

How long would it take to load a 640 × 480 frame buffer with 12 bits per pixel, if 10^5 bits can be transferred per second? How long would it take to load a 24 bits per pixel frame buffer with a resolutions of 1280 by 1024 using this same transfer rate?

Solution: Frame buffer size = 640×480 pixels = $640 \times 480 \times 12$ bits [1 pixel = 12 bits]

Since 10^5 bits takes 1 second

So, $640 \times 480 \times 12$ bits takes $\frac{640 \times 480 \times 12}{10^5} = 36.864$ sec.

Scanned by CamScanner

EXAMPLE 2.10.

Suppose we have a computer with 32 bits per word and a transfer rate of 1 mips (million instructions per second). How long would it take to fill the frame buffer of a 300 dpi (dot per inch) laser printer with a page size of $8\frac{1}{2}$ inch by 11 inches?

Solution: Frame buffer size $= 8\frac{1}{2}$ inch $\times 11$ inch $= 8\frac{1}{2} \times 11$ (inches)^2

$$= \frac{17}{2} \times 11 \times (300)^2 \qquad \text{[1 inch = 300 dpi]}$$

$$= 8415000 \text{ dotes.}$$

Suppose 1 dot = n bits

so, frame buffer size = 8415000 n bits.

Transfer rate = 1 mip = 1×10^6 word per second

$= 32 \times 10^6$ bits per second [1 word = 32 bits]

Time required to fill the frame buffer $= \frac{841500\, n}{32 \times 10^6} = 0.263\ n$ sec.

if $n = 4$ then time taken $= 0.263 \times 4 = 1.052$ sec.

EXAMPLE 2.11.

Consider a non-interlaced raster monitor with a resolution of $n \times m$, a refresh rate of r frames per second, a horizontal ratrace time of t_h and a vertical retrace time of t_v. What is the fraction of the totalm refresh time per frame spent in retrace of the electron beam?

Solution: Since resolution is $n \times m$. So, raster system contains m scan lines and system takes $(m - 1)$ horizontal retraces.

Total time taken for horizontal retrace = $(m - 1) \times$ horizontal retrace time

$= (m - 1) \times t_h$

If there are are r frames then total time $= r \times (m - 1) \times t_h$.

Since, there are r frames then there will be $(r - 1)$ vertical retraces and total time for vertical retrace = $(r - 1) \times$ vertical retrace time $= (r - 1) \times t_v$

$\therefore$ Total time for retrace $= r(m - 1)\ t_h + (r - 1) \times t_v$

Total time taken for one frame $= \frac{r(m-1)\, t_h + (r-1)\, t_v}{r}$ [$\because$ for r frames]

Total refresh time for one frame $= \frac{1}{r}$

The fraction of total refresh time per frame spent in retrace of the electron beam

$$= \frac{\left[\frac{r(m-1)t_h + (r-1)\, t_v}{r}\right]}{\left(\frac{1}{r}\right)} = r(m-1)t_h + (r-1)\ t_v$$

EXAMPLE 2.12.

How much memory is needed for the frame buffer to store a 640 × 400 display 16 gray levels? *(UPTU 2005-06)*

Solution: A shade of gray assign to a pixel. The shades are usually positive integer values taken from the gray scale. In a 8-bit image a gray level can have a value from 0 to 255. 16 gray levels require 4 bits per pixel.

Total number of pixels = 640 × 400 = 256000 pixels

Memory required by frame buffer = 256000 × 4 bits = 1024000 bits

= 128000 bytes = 128 kB.

Summary

The display devices are also known as output devices. The most commonly used output device in a graphics system is a video monitor. The operation of most video monitors is based on CRT. The CRT, invented by Karl Feridand Brown, is an evacuated glass envelope containing a electron gun and a fluorescent screen usually with internal or external means to accelerate and deflect the electrons. In a raster-scan system, the electron beam is swept across the screen, one row at a time from top to bottom. Television sets and printers are examples of systems using raster-scan methods. In random or vector scan display the beam is moved between the end points of the graphics primitives. A DVST stores the picture information as a charge distribution just behind the phosphor-coated screen.

Flat-panel displays are broadly divided into active (light emitting) and passive (light modulating) technologies. Among the active technologies are flat CRTs, plasma-gas discharge electro luminescent etc. LCD and LED are representative of the passive technologies.

Interactive devices are the devices which gives input to the graphics system and provides necessary feedback to the user about the accepted input. Various interactive devices are keyboard, mouse, trackball, joysticks, data glove, touch panels and light pen.

A plotter is a device that draws pictures on paper based on command from a computer. A computer printer produces a hard copy of documents stored in electronic from, usually on physical print media such as paper.

Exercises

1. List the operating characteristics for the following display technologies:
   (*a*) raster systems (*b*) vector systems
   (*c*) plasma panels (*d*) LCDs
2. Write a neat block diagram, explain the architecture of a raster display.
3. What is refresh buffer? Identify the contents and organization of the refresh buffer for the case of raster and vector display.
4. What role does CCD play in an image scanner?
5. What do you understand by VGA and SVGA monitors.
6. What do you understand by Input/Output devices?
7. What is the difference between Raster and Random scan?
8. Explain the working of direct-view storage tubes.
9. Write a short note on
   (*a*) Flat panel display (*b*) Plasma panel display
10. Write a short note on:
    (*a*) Keyboard (*b*) Mouse
    (*c*) Joystick (*d*) Trackball
    (*e*) Digitizer (*f*) Lightpen
    (*g*) Touch panels (*h*) Scanner
    (*i*) Printer

Scanned by CamScanner

$$y = m_2 x + c_2$$

The condition for two straight lines to be parallel is that they have the same slope i.e.,

$$\boxed{m_1 = m_2}$$

Perpendicular straight lines

Two straight lines perpendicular to each other when, the relationship between their slopes is

$$m_1 . m_2 = -1 \text{ or } m_1 = -(1/m_2)$$

That is, the slope of one line should be negative reciprocal of the other, or product of the two slopes should be -1.

5.3. VECTOR GENERATION

The process of "turning on" the pixels for a line segment is called **vector generation**. A scan conversion of line locates the coordinates of the pixels lie on or near an ideal straight line imprinted on 2D raster grid.

Before discussing specific line drawing methods, let us see, what are the characteristics of a line.

1. *The line should appear as a straight line and it should start and end accurately.*
2. *The line should have equal brightness throughout their length.*
3. *The line should be drawn rapidly.*

Even though the rasterization tries to generate a completely straight line, yet in few cases we may not get equal brightness. The brightness of the line is dependent on the orientation of the line. We can observe that the effective spacing between pixels for the 45° line is greater than for the vertical and horizontal lines. This will make the vertical and horizontal lines appear brighter than the 45° line. Complex calculations are required to provide equal brightness along lines of varying length and orientation. Therefore, to draw line some compromises are made such as.

1. Calculate only the approximate line length.
2. Make use of simple arithmetic computations, preferably integer arithmetic.
3. Implement result in hardware or firmware.

The basic idea behind all the drawing algorithms is to reduce the computations and provide the result rapidly. So, most line drawing algorithms use **incremental methods**. In these methods line starts with the starting point. Then a fix increment is added to the current point to get the next point on the line. This is continued till the end of line.

5.4. DIGITAL DIFFERENTIAL ANALYZER (DDA) ALGORITHM

The vector generation algorithms (and curve generation algorithms) which step along the line (or curve) to determine the pixels which should be turned ON are sometimes called **digital differential analyzers** (DDAs). The name comes from the fact that we use the same technique as a numerical method for solving differential equations.

We know that the slope of a straight line is given as

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \quad \ldots(i)$$

This differential equation can be used to obtain a rasterized straight line.

For any given Δx (x interval) along a line, we can compute the Δy (y interval) as

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

i.e.,

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

Similarly, we can obtain the x interval Δx corresponding to Δy as $\Delta x = \frac{\Delta y}{m} = \frac{x_2 - x_1}{y_2 - y_1} \Delta y$

Once the intervals are known, the next values for x and y are obtained as

$$x_{i+1} = x_i + \Delta x$$

$$\boxed{x_{i+1} = x_i + \left(\frac{x_2 - x_1}{y_2 - y_1}\right) \Delta y} \quad \ldots(ii)$$

and

$$y_{i+1} = y_i + \Delta y$$

$$\boxed{y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1} \Delta x} \quad \ldots(iii)$$

For simple DDA either Δx or Δy, whichever is larger, is chosen as some raster unit, *i.e.*,

If $|\Delta x| \geq |\Delta y|$ then

$$\Delta x = 1$$

else

$$\Delta y = 1$$

if $\Delta x = 1$ then

$$y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1} \text{ and}$$

$$x_{i+1} = x_i + 1$$

If $\Delta y = 1$ then

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

$$x_{i+1} = x_i + \frac{x_2 - x_1}{y_2 - y_1}$$

DDA algorithm

1. Read the line end points (x_1, y_1) and (x_2, y_2)
2. $$\Delta x = |x_2 - x_1|$$
   $$\Delta y = |y_2 - y_1|$$
3. If $(\Delta x \geq \Delta y)$ then
   $$\text{length} = \Delta x$$
   else
   $$\text{length} = \Delta y$$
4. Select the raster unit, *i.e.*,

$$\Delta x = \frac{(x_2 - x_1)}{length}$$

$$\Delta y = \frac{(y_2 - y_1)}{length}$$

Scanned by CamScanner

Note: Either Δx or Δy will be one because length is either $x_2 - x_1$ or $y_2 - y_1$. Thus the incremental value for x or y will be one.

5. $x = x_1 + 0.5 * \text{Sign}(\Delta x)$
   $y = y_1 + 0.5 * \text{Sign}(\Delta y)$

Sign function makes the algorithm work in all quadrant. It returns – 1, 0, 1 depending on whether its agreement is < 0, = 0, > 0 respectively. The factor 0.5 makes it possible to round the values in the integer function rather than truncating them.

6. Now plot the points

```
i = 1
while (i ≤ length)
{
    Plot (integer (x), integer (y))
    x = x + Δx
    y = y + Δx
    i = i + 1
}
```

7. Stop.

Advantages of DDA algorithm

1. DDA algorithm is faster than the direct use of the line equation since it calculates points on the line without any floating point multiplication.
2. It is a simplest algorithm and does not require special skills for implementation.

Direct use of the line equation

A simple approach to scan converting a line is to first scan-convert P_1 and P_2 to pixel coordinates (x_1', y_1') and (x_2', y_2') respectively then $m = \dfrac{y_2' - y_1'}{x_2' - x_1'}$ and $c = y_1' - mx_1'$.

If $|m| \le 1$, then for every integer value of x between and excluding x_1' and x_2' calculate the corresponding value of y using equation and scan convert (x, y).

If $|m| > 1$, then for every integer value y between and excluding y_1' and y_2' calculate the corresponding value of x using the equation and scan-convert (x, y).

Disadvantages of DDA algorithm

1. It is orientation dependent, due to this the end point accuracy is poor.
2. A floating-point addition is still needed in determining each successive point which is time consuming. Furthermore, cumulative error due to limited precision in the floating-point representation may cause calculated points to drift away from their true position when the line is relatively long.

EXAMPLE. 5.1

Consider a line from (0, 0) to (4, 6). Use the simple DDA algorithm to rasterize this line.

Solution:

$(x_1, y_1) = (0, 0)$
$(x_2, y_2) = (4, 6)$
$\Delta x = 4 - 0 = 4$
$\Delta y = 6 - 0 = 6$
$\Delta x < \Delta y$ so length $= \Delta y = 6$

$$\therefore \quad \Delta x = \frac{x_2 - x_1}{\text{length}} = \frac{4}{6} = 0.666 \approx 0.667$$

$$\Delta y = \frac{y_2 - y_1}{\text{length}} = \frac{6}{6} = 1$$

$x = 0 + 0.5 \text{ sign } (4/6) = 0.5$
$y = 0 + 0.5 \text{ sign } (1) = 0.5$

Plot (integer (0.5), integer (0.5)) *i.e.*, Plot (1, 1)
$x = x + \Delta x$
$x = 0.5 + 0.667 = 1.167$
$y = 0.5 + 1 = 1.5$

Plot (integer (1.167), integer (1.5)) *i.e.*, Plot (1, 2)
$x = 1.167 + 0.667 = 1.833$
$y = 1.5 + 1 = 2.5$

Plot (integer (1.833), integer (2.5)) *i.e.*, Plot (2, 3)
$x = 1.833 + 0.667 = 2.5$
$y = 2.5 + 1 = 3.5$

Plot (integer (2.5), integer (3.5)) *i.e.*, Plot (3, 4)
$x = 2.5 + 0.667 = 3.167$
$y = 3.5 + 1 = 4.5$

Plot (3, 5)
$x = 3.167 + 0.667 = 4.5$
$y = 4.5 + 1 = 5.5$

The results are shown in the figure below.

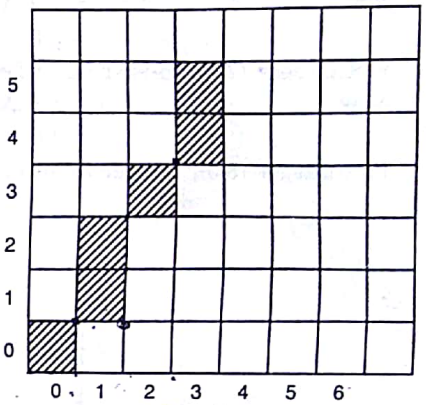


Fig. 5.4.

EXAMPLE 5.2.

Implement the DDA algorithm to draw a line from (0, 0) to (6, 6)

Solution:

$(x_1, y_1) = (0, 0)$
$(x_2, y_2) = (6, 6)$
$\Delta x = x_2 - x_1 = 6 - 0 = 6$
$\Delta y = y_2 - y_1 = 6 - 0 = 6$
$\Delta x = \Delta y$ so length $= \Delta x = 6$

Now the raster unit *i.e.*, $\Delta x = \dfrac{(x_2 - x_1)}{\text{length}} = \dfrac{6}{6} = 1$

and $\Delta y = \dfrac{6}{6} = 1$

$x = x_1 + 0.5 \text{ sign } (1) = 0 + 0.5 \times 1 = 0.5$
$y = y_1 + 0.5 \text{ sign } (+1) = 0 + 0.5 \times 1 = 0.5$

Plot (integer (0.5), integer (0.5)) *i.e.*, **plot (1,1)**

Now $x = x + \Delta x$ and $y = y + \Delta y$
$= 0.5 + 1 = 1.5$ and $= 0.5 + 1 = 1.5$

Plot (integer (1.5), integer (1.5)) *i.e.*, **plot (2, 2)**

Now $x = x + \Delta x$ and $y = y + \Delta y$

Scanned by CamScanner

1.5 + 1 = 2.5 = 1.5 + 1 = 2.5

Plot(integer (2.5), integer (2.5)) *i.e.*, **plot (3, 3)**

Now $x = x + \Delta x$ and $y = y + \Delta y$

$= 2.5 + 1 = 3.5$ $= 2.5 + 1 = 3.5$

Plot(integer (3.5), integer (3.5)) *i.e.*, **plot (4, 4)**

Now $x = x + \Delta x$ and $y = y + \Delta y$

$= 3.5 + 1 = 4.5$ $= 3.5 + 1 = 4.5$

Plot(integer (4.5), integer (4.5)) *i.e.*, **plot (5, 5)**

Now $x = x + \Delta x$ and $y = y + \Delta y$

$= 4.5 + 1 = 5.5$ $= 4.5 + 1 = 5.5$

Plot(integer (5.5), integer (5.5)) *i.e.*, **plot (6, 6)**

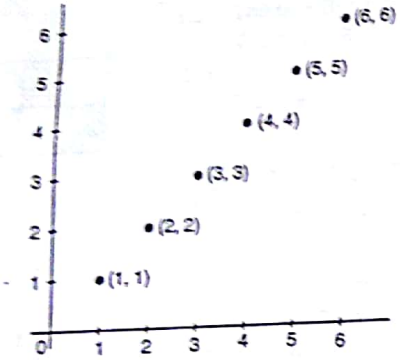


Fig. 5.5.

EXAMPLE 5.3.

Modify the simple straight line DDA to draw either solid, dashed or dotted lines. ***(UPTU-2005-06)***

Solution: DDA algorithm can be modified to make either solid line, dashed line or dotted lines.

Dashed line can be drawn by using DDA algorithm, just leave space after every 4 pixels to make dash.

Dotted line can be drawn by using DDA algorithm. Just leave space after every 1 pixel to make dotted line.

For solid lines

1. Read end points (x_1, y_1) and (x_2, y_2)
2. $\Delta x = |x_2 - x_1|$

   $\Delta y = |y_2 - y_1|$
3. If $(\Delta x \geq \Delta y)$ then

   length = Δx

   else length = Δy

4. $x_{inc} = \dfrac{(x_2 - x_1)}{\text{length}}$

   $y_{inc} = \dfrac{(y_2 - y_1)}{\text{length}}$
5. $x = x_1 + 0.5 * \text{sign}\ (x_{inc})$

   $y = y_1 + 0.5 * \text{sign}\ (y_{inc})$
6. For $i = 1$ to length

```
{
    Plot (integer (x), integer (y))
    x = x + x_inc
    y = y + y_inc
    i = i + 1
}
```

For Dashed lines

1. Read end points (x_1, y_1) and (x_2, y_2)
2. $\Delta x = |x_2 - x_1|$

   $\Delta y = |y_2 - y_1|$
3. If $(\Delta x \geq \Delta y)$ then

   length = Δx

   else length = Δy
4. $x_{inc} = \dfrac{(x_2 - x_1)}{\text{length}}$

   $y_{inc} = \dfrac{(y_2 - y_1)}{\text{length}}$
5. $x = x_1 + 0.5 * \text{sign}\ (x_{inc})$

   $y = y_1 + 0.5 * \text{sign}\ (y_{inc})$
6. $k = 0$

   for $i = 1$ to length

```
{
    if (k < = 3)
    {
        x = x + x_inc
        y = y + y_inc
        k = k + 1
    }
    else
    {
        x = x + x_inc
        y = y + y_inc
        k = 0
```

Advantages of DDA:

01) This is very faster algo

02) This does not require special techniques

Disadvantages of DDA:

01) It is orientation dependent

02) Floating - point addition is still needed which is time - consuming

Scanned by CamScanner

```
    }
    Plot (integer (x), integer (y))
}
```

Dotted line

1. Read end points (x_1, y_1) and (x_2, y_2)
2. $\Delta x = |x_2 - x_1|$

   $\Delta y = |y_2 - y_1|$
3. If $(\Delta x \geq \Delta y)$ then

   length = Δx

   else length = Δy
4. $x_{inc} = \dfrac{(x_2 - x_1)}{\text{length}}$

   $y_{inc} = \dfrac{(y_2 - y_1)}{\text{length}}$
5. $x = x_1 + 0.5 * \text{sign}(x_{inc})$

   $y = y_1 + 0.5 * \text{sign}(y_{inc})$
6. $k = 0$

   for $i = 1$ to length

```
{
    if (k < 1)
    {
        x = x + x_inc
        y = y + y_inc
        k = k + 1
    }
    else
    {
        x = x + x_inc
        y = y + y_inc
        k = 0
    }
    Plot (integer (x), integer (y))
}
```

5.5. BRESENHAM'S LINE ALGORITHM *(UPTU, 2008-09)*

One serious drawback of the DDA algorithm is that it is very time consuming as it deals with a rounding off operation and floating point arithmetic. The algorithm developed by Bresenham is more accurate and efficient compared to DDA algorithm because it cleverly avoids the "Round" function and scan converts line using only incremental integer calculation.

Bresenham's line algorithm uses only integer addition and subtraction and multiplication by 2, and we know that the compute can perform the operations of integer addition and subtraction very rapidly. The computer is also time-efficient when performing integer multiplication by powers of 2.

The basic principle of Bresenham's line algorithm is to find the optimum raster locations to represent the straight lines. To accomplish this the algorithm always increments either x or y by one unit depending on the slope of line. Once this is done, then increment in other variable is found on the basis of the distance between the actual line location and the nearest pixel. This distance is called **decision variable** or the **error term**.

Figure illustrated sections of a display screen where straight line segments are to be drawn. The vertical axis show scan-line, positions and the horizontal axis identify pixel columns.

Sampling at unit x intervals in these examples, we need to decide which of two possible pixel positions is closer to the line path at each sample step. Starting from the left endpoint shown in figure 5.6, we need to determine at the next sample position whether to plot the pixel at position (11, 11) or the one at (11, 12). Similarly, Figure 5.6(*b*) shown a negative slop line path starting from the left endpoint at pixel position (50, 50). In this one, do we select the next pixel position as (51, 50) or as (51, 49)? These questions are answered with Bresenham's line algorithm by testing the sign of an integer parameter, whose value is proportional to the difference between the separations of the two pixel positions from the actual line path.

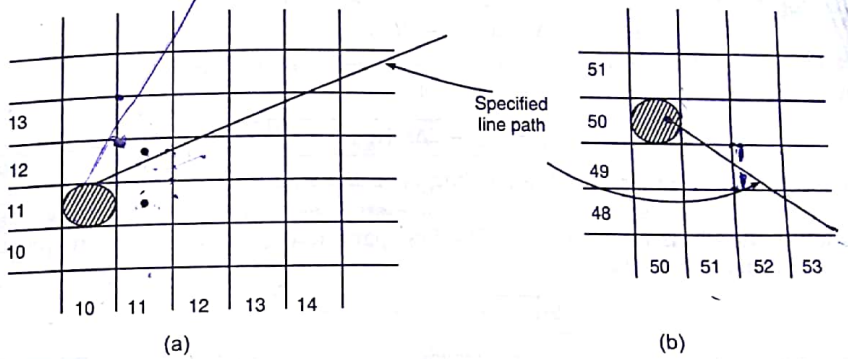


Fig. 5.6.

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 the left end point (x_0, y_0) of a given line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path Figure demonstrates the kth step in this process. 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 $(x_k + 1, y_k + 1)$.

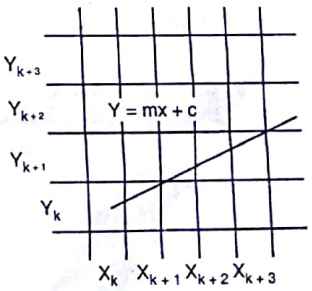


Fig. 5.7.

At sampling position $x_k + 1$, we label vertical pixel separations from the mathematical line path as d_1 and d_2. The y coordinate on the mathematical line at pixel column position $x_k + 1$ is calculated as

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

Scanned by CamScanner

Then, $d_1 = y - y_k$

$= m(x_k + 1) + c - y_k$

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

$= y_k + 1 - m(x_k + 1) - c$

The difference between these two separations is

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

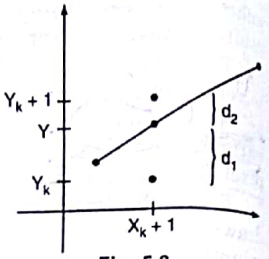


Fig. 5.8.

A decision parameter p_k for the kth step in the line algorithm can be obtained by rearranging equation so that it involves only integer calculations. We accomplish this by substituting $m = \Delta y/\Delta x$, where Δy and Δx are the vertical and horizontal separations of the endpoint positions, and defining :

$$p_k = \Delta x\,(d_1 - d_2) = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + d$$

The sign of p_k is the same as the sign of $d_1 - d_2$, since $\Delta x > 0$ for our example. Parameter d is constant and has the value $2\Delta y + \Delta x(2c - 1)$, which is independent of pixel position and will be eliminated in the recursive calculations for p_k.

If pixel at y_k is closer to the line path than the pixel at $y_k + 1$ that means $d_1 < d_2$ *i.e.*, decisions parameters p_k is – ve. In that case we plot the lower pixel, otherwise, we plot the upper pixel.

At step $k + 1$, the decision parameters is evaluated as

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

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

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

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

where $y_{k+1} - y_k$ is either 0 or 1 depending upon the sign of parameter p_k.

This recursive calculation of decision parameters is performed at each integer x starting at the left and co-ordinate of the line. The first parameter p_0 is evaluated at the starting point (x_0, y_0) and $m = \Delta y/\Delta x$ as

$$\boxed{p_0 = 2\Delta y - \Delta x.}$$

We can summarize Bresenham line drawing for a line with a positive slope less than 1 in the following listed steps. The constants $2\Delta y$ and $2\Delta y - 2\Delta x$ are calculated once for each line to be scan converted, so the arithmetic involves only integer addition and substraction of these two constants.

Bresenham's Line-drawing algorithm for $|m| < 1$

1. Input the two line endpoints store that 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

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

   where $\Delta x = |x_2 - x_1|$ and $\Delta y = |y_2 - y_1|$
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.

EXAMPLE. 5.4.

Consider the line from (5, 5) to (13, 9). Use the Bresenham algorithm to rasterize the line.

Solution: $(x_1, y_1) = (5, 5)$ $(x_2, y_2) = (13, 9)$

$\Delta x = 13 - 5 = 8$

$\Delta y = 9 - 5 = 4$

$$|m| = \frac{4}{8} = \frac{1}{2} < 1$$

$$p_0 = 2\Delta y - \Delta x = 2 \times 4 - 8 = 0$$

So, the next point is (6, 6) and

$$p_1 = p_0 + 2\Delta y - 2\Delta x = 0 + 2 \times 4 - 2 \times 8 = 8 - 16 = -8$$

$p_1 < 0$, so the next point to plot is (7, 6)

and $$p_2 = p_1 + 2\Delta y = -8 + 2 \times 4 = 0$$

So, the next point to plot is (8, 7) and

$$p_3 = p_2 + 2\Delta y - 2\Delta x = 0 + 2 \times 4 - 2 \times 8 = 8 - 16 = -8$$

$p_3 < 0$, so the next point to plot is (9, 7) and

$$p_4 = p_3 + 2\Delta y = -8 + 2 \times 4 = 0$$

and the next point to plot is (10, 8) and

$$p_5 = p_4 + 2\Delta y - 2\Delta x = 0 + 2 \times 4 - 2 \times 8 = -8$$

$p_5 < 0$, so the next point to plot is (11, 8) and

$$p_6 = p_5 + 2\Delta y = -8 + 2 \times 4 = 0$$

and the next point to plot is (12, 9) and

$$p_7 = p_6 + 2\Delta y - 2\Delta x = 0 + 2 \times 4 - 2 \times 8 = 8 - 16 = -8$$

$p_7 < 0$ so, the next point to plot is (13, 9)

The results are plotted as shown in figure below

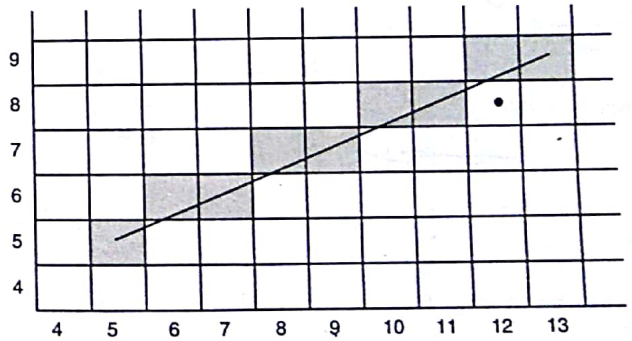


Fig. 5.9.

This method is well suited for the line having the slope less than 45° *i.e.*, first octant. We can have the modification in these steps to draw the line in any quadrant. Such algorithm can easily developed by considering the quadrant in which the line lies and the stope. When the magnitude of the slope is greater than 1, y increment by one and the decision variable

Scanned by CamScanner

is use to determine when to increment x. The x and y incremental values depend in the quadrant in which the line exists. This is shown in Fig. 5.10.

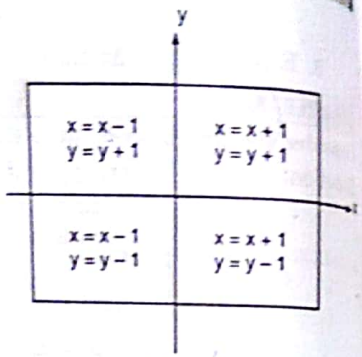


Fig. 5.10.

Generalized Bresenham's Algorithm

1. Read the line end point (x_1, y_1) and (x_2, y_2) such that they are not equal
2. $\Delta x = |x_2 - x_1|$
   $\Delta y = |y_2 - y_1|$
3. Initialize starting point $x = x_1$
   $y = y_1$
4. $S_1 = \text{Sign}(x_2 - x_1)$.
   $S_2 = \text{Sign}(y_2 - y_1)$

[Sign function returns – 1, 0, 1 depending on whether its agreement < 0, = 0, > 0 respectively].

5. If $\Delta y > \Delta x$ then
   exchange Δx & Δy
   ex_change = 1
   else
   ex_change = 0
6. $e = 2 * \Delta y - \Delta x$
7. $i = 1$
8. plot (x, y)
9. while $(e \geq 0)$
   {
   if (ex_change = 1) then
   $x = x + s_1$
   else
   $y = y + s_2$
   $e = e - 2 * \Delta x$
   }
10. If ex_change = 1 then
    $y = y + s_2$
    else
    $x = x + s_1$
    $e = e + 2 * \Delta y$
11. $i = i + 1$
12. If $(i \leq \Delta x)$ then go to step (8)
13. stop.

EXAMPLE. 5.5.

Consider a line from (0, 0) to (6, 7). Use Bresenham's algorithm to rasterize the line.

Solution: $(x_1, y_1) = (0, 0)$ $(x_2\ y_2) = (6, 7)$

$x = 0$ $\Delta x = |x_2 - x_1| = 6$

$y = 0$ $\Delta y = |y_2 - y_1| = 7$

$\Delta y > \Delta x$ so exchange Δx and Δy and

$\Delta x = 7$

Now $\Delta y = 6$

plot (0, 0)

$e = 2 * \Delta y - \Delta x$

$= 2 \times 6 - 7 = 5$ *i.e.*, $e = 5$ $\quad \begin{cases} S_1 = 1 \\ S_2 = 1 \end{cases}$

$e > 0$ then $x = x + S_1 = 0 + 1 = 1$

and $e = e - 2 * \Delta x = 5 - 2 * 7 = 5 - 14 = -9$

$e < 0$ so exit while loop.

Here ex_change = 1 then $y = y + S_2$

$y = 0 + 1 = 1$ $\quad y = 1$

and $e = e + 2 * \Delta y = -9 + 2 * 6$

$= -9 + 12 = 3$, *i.e.*, $e = 3$

Now plot (x, y) *i.e.*, (1, 1)

Now $e = 3$ *i.e.*, $e \geq 0$

and $x = x + S_1 = 1 + 1$

$x = 2$

$e = e - 2 * \Delta x = 3 - 2 * 7 = 3 - 14 = -11$

$y = y + S_2 = 1 + 1 = 2$

$e = e + 2 * \Delta y = -11 + 2 * 6 = 1$

$e = 1$

Plot (x, y) *i.e.*, (2, 2).

Now $e = 1$ i.e., $e \geq 0$

$x = 2 + 1 = 3$

$e = e - 2 * \Delta x = 1 - 2 * 7 = -13$

$y = 2 + 1 = 3$

$e = e + 2 * \Delta y = -13 + 2 * 6 = -1$

$e = -1$

Plot (x, y) *i.e.*, (3, 3)

Now $e = -1$ *i.e.*, $e < 0$

$y = y + S_2$

$y = 3 + 1 = 4$

$e = e + 2 * \Delta y = -1 + 2 * 6 = 11$

Plot (x, y) *i.e.*, (3, 4)

Now $e = 11$ *i.e.*, $e \geq 0$

$x = 3 + 1 = 4$

$e = e - 2 * \Delta x = 11 - 2 * 7 = -3$

$y = 4 + 1 = 5$

$e = e + 2 * \Delta y = -3 + 2 * 6 = 9$

Plot (x, y) *i.e.*, (4, 5)

Now $e = 9$ *i.e.*, $e \geq 0$

$x = 4 + 1 = 5$

$e = e - 2 * \Delta x = 9 - 2 * 7 = -5$

$y = 5 + 1 = 6$

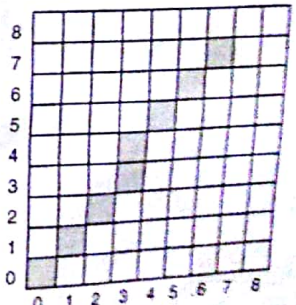


Fig. 5.11.

Scanned by CamScanner

$e = e + 2 * \Delta y = -5 + 2 * 6 = 7$

Plot (x, y) *i.e.*, (5, 6)

Now $e = 7$ *i.e.*, $e \geq 0$

$x = 5 + 1 = 6$

$e = e - 2 * \Delta x = 7 - 2 * 7 = -7$

$y = 6 + 1 = 7$

$e = e + 2 * \Delta y = -7 + 2 * 6 = 5$

Plot (6, 7)

Thus the pixels to be plotted are (0, 0) (1, 1) (2, 2) (3, 3) (3,4) (4, 5) (5, 6) (6, 7).

5.6. CIRCLE-GENERATING ALGORITHMS

5.6.1. Basic Concepts in Circle Drawing

A circle is a symmetrical figure. It has eight-way symmetry *i.e.*, the shape of the circle is similar in each quadrant as shown in the figure.

Thus, any circle generating algorithm can take advantage of the circle symmetry to plot eight points by calculating the co-ordinates of any one point. For example, calculation of a circle point (x, y) in one octant yields the circle points shown for the other seven octants just by reflection. Taking advantage of the circle symmetry in this way, we can generate all pixel positions around a circle by calculating only the points within the sector from $x = 0$ to $x = y$.

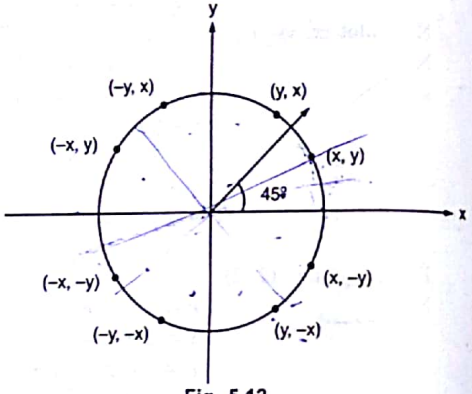


Fig. 5.12.

Representation of Circles

There are two standard methods of mathematically representing a circle centered at the origin.

(*a*) Polynomial method

(*b*) Trigonometric method

Polynomial Method. In this method, circle is represented by a polynomial equation

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

where x is the x co-ordinate
y is the y co-ordinate
r is the radius of circle

Polynomial equation can be used to find y co-ordinate for the known x co-ordinate. Therefore, the scan converting circle using polynomial method is achieved by stepping x from 0 to $r\sqrt{2}$ and each y co-ordinate is found by evaluating $\sqrt{r^2 - x^2}$ for each step of x. This generates the 1/8 portion of the circle. Remaining part of the circle can be generated just be reflection.

Trigonometric method. In this method, the circle is represented by use of trigonometric functions

$x = r \cos \theta$

$y = r \sin \theta$

where $x = x$ co-ordinate θ = current angle

$y = y$ co-ordinate r = radius of the circle

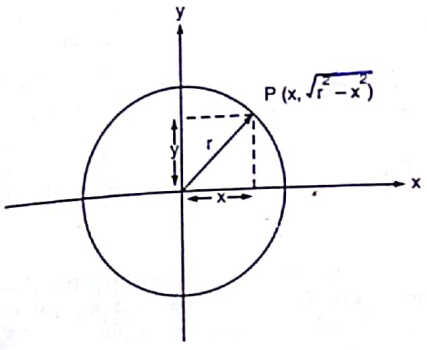


Fig. 5.13.

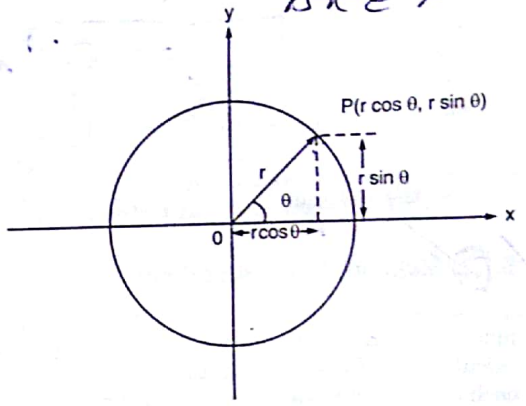


Fig. 5.14.

The scan converting circle using trigonometric method is achieved by stepping θ from 0 to $\pi/4$ radians, and each value of x and y is calculated.

This method is more inefficient than the polynomial method because the computation of the values of sin θ and cos θ is even more time consuming than the calculations required in the polynomial method.

5.7. CIRCLE DRAWING ALGORITHM

5.7.1. DDA Circle Drawing Algorithm

We know the equation of circle, whose center is (0, 0) is given as $x^2 + y^2 = r^2$.

The DDA can be used to draw the circle by defining circle as a differential equation.

So, $2x\,dx + 2y\,dy = 0$

or $x\,dx + y\,dy = 0$

or $y\,dy = -x\,dx$

or $$\frac{dy}{dx} = -\frac{x}{y}$$

We can construct the circle by using incremental x value $\Delta x = \epsilon y$ and incremental y value $\Delta y = -\epsilon x$, where ϵ is calculated from the radius of the circle as $\epsilon = 2^{-k}$. Apply these incremental steps, we get

$$x_{n+1} = x_n + \epsilon y_n$$
$$y_{n+1} = y_n - \epsilon x_n$$

Representing this in matrix form, we have

Scanned by CamScanner

$$[x_{n+1}, y_{n+1}] = [x_n \;\; y_n] \begin{bmatrix} 1 & -\epsilon \\ \epsilon & 1 \end{bmatrix}$$

To get the circle should be closed, we have to make one correction in the equation. We have to replace x_n by x_{n+1} in the equation of y_{n+1}

Therefore now

$$x_{n+1} = x_n + \epsilon y_n$$

$$y_{n+1} = y_n - \epsilon x_{n+1}$$

i.e.,

$$y_{n+1} = (1 - \epsilon^2)\, y_n - \epsilon x_n$$

so,

$$[x_{n+1} \;\; y_{n+1}] = [x_n \;\; y_n] \begin{bmatrix} 1 & -\epsilon \\ \epsilon & 1-\epsilon^2 \end{bmatrix}$$

and the value of $\begin{bmatrix} 1 & -\epsilon \\ \epsilon & 1-\epsilon^2 \end{bmatrix}$ is 1.

5.7.2 Midpoint Circle Algorithm

As in the raster line algorithm, we sample at unit intervals and determine the closest pixel position to the specified circle path at each step. We can set up our algorithm to calculate pixel positions around a circle path centered at the co-ordinate origin (0, 0). Then each calculated position (x, y) is moved to its proper screen position by adding x_c to x and y_c to y, where (x_c, y_c) is the actual centre of the circle with radius r.

Only one octant of the circle needs to be generated. The other parts can be obtained by successive reflections. If the first octant (0 to 45°) is generated the second octant can be obtained by reflection through the line $y = x$ to yield the first quadrant. The results in the first quadrant are reflected through line $x = 0$, to obtain those in the second quadrant. The combined results on the upper semicircle are reflected through the line $y = 0$ to complete the circle.

To apply the midpoint method, we define the equation of a circle with (0, 0) as its centre as

$$F_{circle}(x, y) = x^2 + y^2 - r^2$$

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

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

Thus, the circle function is the decision parameter in the midpoint algorithm and we can set up incremental calculations for this function as we did in the line algorithm.

In Fig. 5.15 we have plotted the pixel at (x_k, y_k), we next need to determine whether the pixel at position (x_{k+1}, y_k) or one at position (x_{k+1}, y_{k-1}) or at position (x_k, y_{k-1}) is closer to the circle.

Firstly, we check that if the diagonal point is inside or outside the circle *i.e.*,

$\Delta = (x_{k+1})^2 + (y_{k-1})^2 - r^2$ is < 0, = 0 or > 0

If $\boxed{\Delta < 0}$, then the diagonal point is inside the circle *i.e.*, we have to select one of the two point (x_{k+1}, y_k) or (x_{k+1}, y_{k-1}), the one which is more closer to the circle boundary. For this, we check for the midpoint between these two points *i.e.*,

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

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

If $p_k > 0$, this midpoint is outside the circle, and the pixel on the scan line y_{k-1} is closer to the circle boundary otherwise the mid position is inside or on the circle boundary and we select the pixel on scan line y_k.

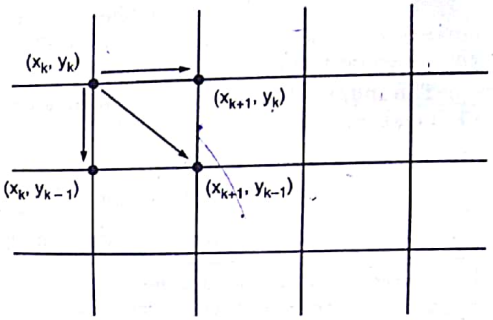


Fig. 5.15.

Similarly, if $\boxed{\Delta > 0}$, then the diagonal point is outside the circles, *i.e.*, we have to select one of the two points (x_{k+1}, y_{k-1}) and (x_k, y_{k-1}) the one which is more closer to the circle boundary. For this, once again we check for the midpoint between these two points *i.e.*,

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

If $p_k > 0$, this midpoint is outside the circle and the pixel (x_k, y_{k-1}) is selected, otherwise the midpoint is inside or on the circle boundary ($p_k < 0$) and we select the pixel (x_{k+1}, y_{k-1})

If $\boxed{\Delta = 0}$, pixel (x_{k+1}, y_{k-1}) is selected.

The initial decision parameter is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$

$$p_0 = f_{circle}\left(1, r - \frac{1}{2}\right) = 1 + \left(r - \frac{1}{2}\right)^2 - r^2 = \frac{5}{4} - r$$

if the radius r is specified as an integer, we can simply round p_0 as $p_0 = 1 - r$.

Successive decision parameters are obtained using incremental calculations. We obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position $x_{k+1} + 1 = x_k + 2$.

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

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

where y_{k+1} is either y_k or y_{k-1}, depending on the sign of p_k.

If p_k is negative (– ve)

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

$$p_{k+1} = p_k + 2(x_k + 1) + 1 = p_k + 2x_{k+1} + 1$$

If p_k is positive (+ ve)

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

$$\therefore \quad p_{k+1} = p_k + 2(x_k + 1) + 1 - 2y_{k+1}$$

Thus, increments for obtaining p_{k+1} are either $2x_{k+1} + 1$ (if p_k is – ve) or $2x_{k+1} + 1 - 2y_{k+1}$ (if p_k is + ve) and the evaluation of the terms $2x_{k+1}$ and $2y_{k+1}$ can be done as

$$2x_{k+1} = 2x_k + 2$$

$$2y_{k+1} = 2y_k - 2$$

Scanned by CamScanner

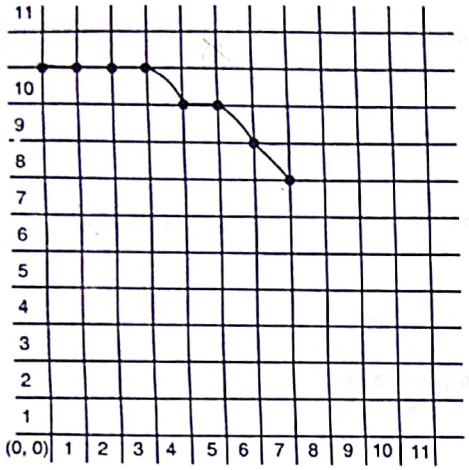


Fig. 5.16.

5.7.3 Bresenham's Circle Drawing Algorithm

Bresenham's method of drawing the circle is an efficient method because it avoids trigonometric and square root calculation by adopting only integer operation involving squares of the pixel separation distances.

The Bresenham's circle drawing algorithm considers the eight way symmetry of the circle. It plots 1/8th part of the circle from 90° to 45°. As circle is drawn from 90° to 45°, the x-moves in + ve direction and y moves in the – ve direction.

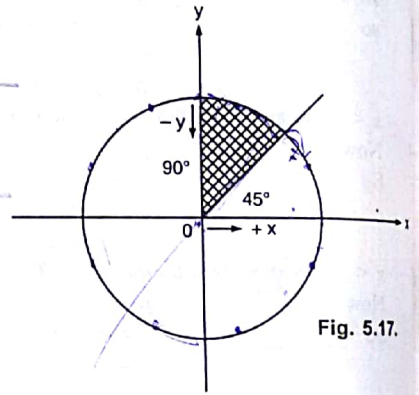


Fig. 5.17.

To achieve best approximation to the true circle we have to select those pixels in the raster that falls the least distance from the true circle, Let us observe the 90° to 45° portion of the circle, each new point closest to the true circle can be found by applying either of two options :

(*a*) Increment in positive x direction by one unit or

(*b*) Increment in positive x direction and negative y direction both by one unit.

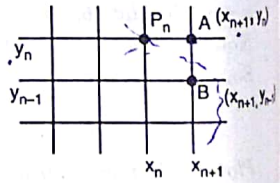


Fig. 5.18.

If P_n is a current point with co-ordinates (x_n, y_n) then the next point could be either A or B.

We have to select A or B, depending on which is close to the circle, and for that we have to perform some test.

The closer pixel amongst these two can be determined as follows.

The distances of pixels A and B from the origin (0, 0) are given by

$$d_A = \sqrt{(x_{n+1} - 0)^2 + (y_n - 0)^2}$$

$$d_A = \sqrt{x_{n+1}^2 + y_n^2}$$

and

$$d_B = \sqrt{x_{n+1}^2 + y_{n-1}^2}$$

Now, the distances of pixels A and B from the true circle whose radius r are given as

$$\delta_A = d_A - r \text{ and } \delta_B = d_B - r$$

To avoid square root in derivation of decision variable, we use

$$\delta_A = d_A^2 - r^2 \text{ and } \delta_B = d_B^2 - r^2$$

But δ_A is always positive and δ_B is always negative. Therefore we can define decision variable d_i as

$$\boxed{d_i = \delta_A + \delta_B}$$

If $d_i < 0$ *i.e.*, $\delta_A < \delta_B$ then only x is incremented.

Otherwise x is incremented in positive direction and y is incremented in negative direction *i.e.*,

For $d_i < 0$

$$x_{i+1} = x_i + 1$$

else

$$x_{i+1} = x_i + 1$$

$$y_{i+1} = y_i - 1$$

The equation for d_i at starting point *i.e.*, $x = 0$ and $y = r$ is as follows

$$\begin{aligned} d_i = \delta_A + \delta_B &= x_{n+1}^2 + y_n^2 - r^2 + x_{n+1}^2 + y_{n-1}^2 - r^2 \\ &= (0+1)^2 + r^2 - r^2 + (0+1)^2 + (r-1)^2 - r^2 \\ &= 1 + 1 + r^2 + 1 - 2r - r^2 = 3 - 2r \end{aligned}$$

As stated earlier, if $d_i < 0$ then it implies A is closer to the true circle, hence increment only x value.

i.e., If $d_i < 0$ then $x_{i+1} = x_i + 1$

else

$$x_{i+1} = x_i + 1$$

$$y_{i+1} = y_i - 1$$

So, recompute the new decision value, by substituting new value of x_i we get

$$d_{i+1} = d_i + 4x_i + 6 \quad \text{if } d_i < 0$$

$$d_{i+1} = d_i + 4(x_i - y_i) + 10 \quad \text{if } d_i \geq 0$$

Algorithm for Bresenham's circle drawing

1. Read the radius r of the circle.
2. Initialize the decision variable
   $d = 3 - 2 * r$
3. $x = 0$
   $y = r$
4. If we are using octant symmetry to plot the pixels then until $(x < y)$ we have to perform following steps.
   if $(d < 0)$ then
   $d = d + 4x + 6$
   $x = x + 1$
   else
   $y = y$

Scanned by CamScanner

$d = d + 4(x - y) + 10$
$y = y - 1$
$x = x + 1$

5. Plot (x, y)
6. Determine the symmetry points in other octants also.
7. Stop.

EXAMPLE. 5.7.

Plot a circle by Bresenham's algorithm whose radius is 3 and center is (0, 0).

Solution:

$r = 3$
$d = 3 - 2 \times 3 = -3$
$d = -3$
$x = 0$
$y = 3$

Plot first point (x, y) i.e., $(0, 3)$

Now

$d = -3\ (< 0)$ then
$d = -3 + 4 \times 0 + 6 = 3$
$x = x + 1 = 0 + 1 = 1$
$y = y = 3$

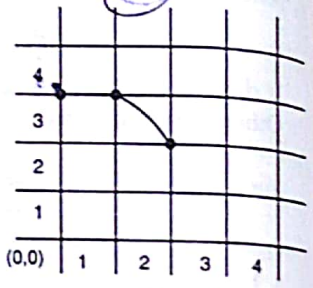


Fig. 5.19.

Plot (1, 3)

Now

$d = 3 (> 0)$ then
$d = d + 4(x - y) + 10$
$= 3 + 4\ (1 - 3) + 10$
$= 3 - 8 + 10 = 5$
$x = x + 1 = 1 + 1 = 2$
$y = y - 1 = 3 - 1 = 2$

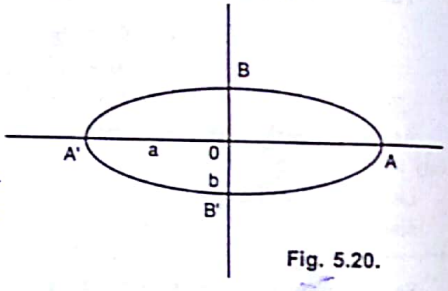


Fig. 5.20.

Plot (2, 2)

After plotting these points, we will get the circle.

5.8. BASIC CONCEPTS IN ELLIPSE

The general equation for an ellipse with major axis $2a$ and minor axis $2b$, centered at the origin is

$$\boxed{(x/a)^2 + (y/b)^2 = 1}$$

where $2a$ = length of major axis
$2b$ = length of minor axis

$$b^2x^2 + a^2y^2 - a^2b^2 = 0 \qquad ...(a)$$

For any given value of x, say x_p it will give two values of y such as

$$y = \pm\, b\ \sqrt{1 - x^2/a^2}$$

and for any given value of y, say y_p it will give two values of x such as

$$x = \pm\, a\ \sqrt{1 - y^2/b^2}$$

Parametric equations for an Ellipse

The parametric form of the equations for the ellipse are

$$x = a\ .\ \cos\theta$$
$$y = b\ .\ \sin\theta$$

where θ ranges form 0 to 2π.

Now, $y/x = \dfrac{b.\sin\theta}{a\cos\theta}$ or $\tan\alpha = \dfrac{b}{a}\ \tan\theta$.

From co-ordinate geometry, we get

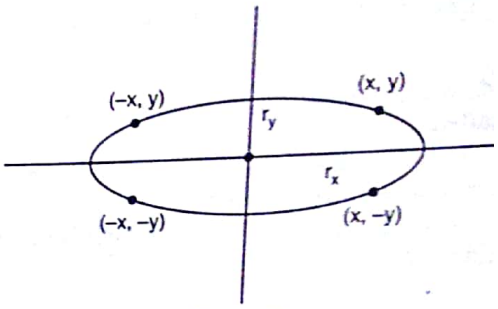


Fig. 5.21.

$$f(x, y) = b^2x^2 + a^2y^2 - a^2b^2 \begin{cases} < 0 \text{ implies } (x,y) \text{ inside the ellipse} \\ = 0 \text{ implies } (x,y) \text{ on the ellipse} \\ > 0 \text{ implies } (x,y) \text{ outside the ellipse} \end{cases}$$

An ellipse in standard position is symmetric between quadrants but unlike a circle, it is not symmetric between the two octants of a quadrant. Thus we have to calculate pixel positions along the elliptical arc throughout one quadrant, then we obtain positions in the remaining 3 quadrants by using symmetry property (see Fig. 5.21).

where r_x represents semimajor axis
r_y represents semiminor axis

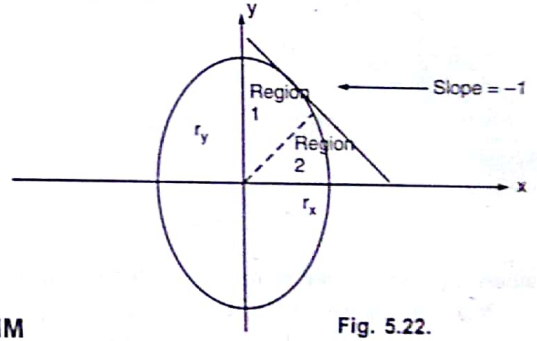


Fig. 5.22.

5.9. ELLIPSE GENERATING ALGORITHM

An ellipse is an elongated circle. Therefore, elliptical curve can be generated by modifying circle-drawn procedures.

Midpoint Ellipse Algorithm

The midpoint ellipse method is applied throughout the first quadrant in two parts *i.e.*, region 1 and region 2. We are forming these regions by considering the slope of the curve. If the slope of the curve is less than – 1 then we are in region 1 and when the slope becomes greater than – 1 then in region 2.

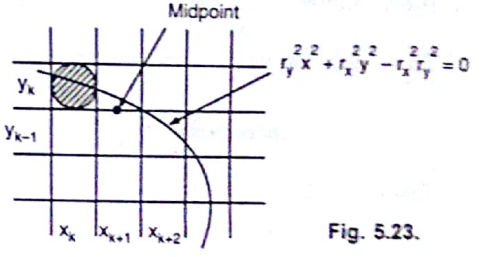


Fig. 5.23.

Starting at $(0, r_y)$ we have to take unit steps in the x direction until we reach the boundary between region 1 and region 2. Then we have to switch to unit steps in the y direction over the remainder of the curve in the first quadrant. At each step, we need to test the value of the slope of curve. The slope of the ellipse is given as

$$\frac{dy}{dx} = -\frac{2r_y^2x}{2r_x^2y} \quad (\because f(x, y) = r_y^2x^2 + r_x^2y^2 - r_x^2r_y^2)$$

At the boundary between region 1 and region 2

$$\frac{dy}{dx} = -1 \quad \text{and} \quad 2r_y^2\ x = 2r_x^2y$$

Scanned by CamScanner

2-D TRANSFORMATION

Chapter Outline

- Introduction
- Representation of Points or Objects in Matrix Form
- Geometric Transformations
- Combined Transformations
- Affine Transformation
- Display Procedures
- Homogeneous Coordinates
- Transformation Between co-ordinate Systems
- Transformation Routines

8.1. INTRODUCTION

It is essential for a graphics system to allow the user to change the way objects appear. In the real world we frequently rearrange objects or look at them from different angles. The effects that we desire include changing the size of the object, its position on the screen, or its orientation. The implementation of such a change is called *transformation*. The term 'transform' means 'to change'. This change can either be of shape, size or position of the object. To perform transformation on any object, object matrix 'X' is multiplied by the transformation matrix T.

$$\begin{bmatrix}\text{Transformed object}\\ \text{matrix}\end{bmatrix} = \begin{bmatrix}\text{Original object}\\ \text{matrix}\end{bmatrix} \times \begin{bmatrix}\text{Transformation}\\ \text{matrix}\end{bmatrix} \qquad [X^*] = [X][T]$$

or $$[X^*] = [X]\ [T]$$

The various transformations possible on an object are as follows:

- Rotation
- Scaling
- Shearing etc.
- Reflection
- Translation

Each of the above transformation is carried out by using a different transformation matrix.

The fundamental objective of 2-D transformation is to simulate the movement and manipulation of objects in the plane. Points and lines which join them, along with appropriate drawing algorithm are used to represent objects. The ability to transform these points and lines is achieved by translation, rotating, scaling and reflection. Two points of view are used

Scanned by CamScanner

for describing the object movement. The first is that the object itself is moved relative to a stationary co-ordinate system or background. The mathematical statement of this view point is described by *geometric transformations* applied to each point of the object. The second point of view holds that the object is held stationary, while the co-ordinate system is moved relative to the object. This effect is described by *co-ordinate transformation.*

Thus transformation is described in two categories:

(a) Geometric transformation

(b) Co-ordinate transformation

The transformations are used directly by application programs and within many graphic subroutines.

8.2. REPRESENTATION OF POINTS OR OBJECTS IN MATRIX FORM

In 2D co-ordinate system, any point is represented in terms of x and y co-ordinates. The point (x, y) can be converted into matrix in the following two ways:

(a) Row-major matrix

$$[x \quad y]_{1\times 2}$$

(b) Column-major matrix

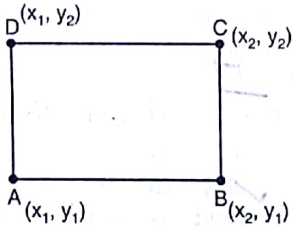
$$\begin{bmatrix} x \\ y \end{bmatrix}_{2\times 1}$$

The above two matrices are frequently called *position vectors*. A series of points, each of which is a position vector relative to some co-ordinate system, is stored in a computer as a matrix or array of numbers. The position of these points is controlled by manipulating the matrix which defines the points. Lines are drawn between the points to generate lines, curves or pictures.

Suppose, we represent a rectangle in matrix form. Let (x_1, y_1) and (x_2, y_2) be the opposite vertices of a rectangle. Then, the four vertices of rectangle will be: $(x_1\ y_1)$, $(x_2\ y_1)$, $(x_2\ y_2)$, (x_1, y_2).

In order to represent this rectangle in matrix form as

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_1 \\ x_2 & y_2 \\ x_1 & y_2 \end{bmatrix}$$



8.3. GEOMETRIC TRANSFORMATIONS

Changes in orientation, size and shape are accomplished with geometric transformations that alter the co-ordinate descriptions of objects. The basic geometric transformations are translation, rotation and scaling. Other transformations that are often applied to objects include reflection and shear.

8.3.1. Translation *(UPTU 2009)*

Translation consists of a shift of the object parallel to itself in any direction in the (x, y) plane. Any such shift can be accomplished by a shift in x-direction plus a shift in y-direction.

If the amount of x-shift is called t_x and the amount of y-shift t_y, the translation of the point (x, y) into the point (x', y') is expressed by the formulas.

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

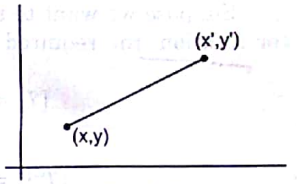


Fig. 8.1.

To translate an object with multiple points, we just translate each point individually and connect them together.

Translation is a rigid-body transformation that moves objects without deformation. That is, every point on the object is translated by the same amount. Translation is the only transformation that is not in relation to a reference point. Its effect is independent of the original position of the object.

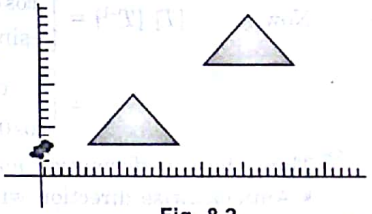


Fig. 8.2.

8.3.2. Rotation *(UTPU 2009)*

A two-dimensional rotation is applied to an object by repositioning it along a circular path in the xy plane. Points can be rotated through an angle θ about the origin. The sign of angle determines the direction of rotation. Positive values for the rotation angle defines counterclockwise rotations and negative values rotate objects in clockwise directions.

Suppose rotation by θ transforms the point $P(x, y)$ into $P'(x', y')$. Because the rotation is about the origin, the distances from the origin to P and to P' is r are equal.

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

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

Put the values of x and y, we get

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

In matrix form (Row-major order)

$$[X'] = [X]\ [T] \text{ or } [x'\ \ y'] = [x\ \ y]\ [T]$$

Put the value of x' and y', we get

$$[x\cos\theta - y\sin\theta \quad \sin\theta + y\cos\theta] = [x \quad y]\ [T]$$

Hence
$$[T] = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

$$|T| = \cos^2\theta + \sin^2\theta = 1.$$

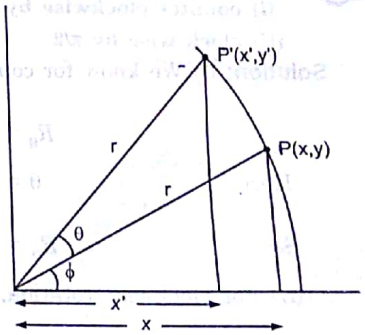


Fig. 8.3.

Note: Determinant of a pure rotation matrix is always + 1.

In column-major order

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

Scanned by CamScanner

Suppose we want to rotate P' back to point P *i.e.*, to perform the inverse transformation or rotation, the required angle is $-\theta$. then,

$$[T] = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

$$[T^{-1}] = \begin{bmatrix} \cos(-\theta) & \sin(-\theta) \\ -\sin(-\theta) & \cos(-\theta) \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Now
$$[T]\,[T^{-1}] = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} \cos^2\theta+\sin^2\theta & -\cos\theta\sin\theta+\sin\theta\cos\theta \\ \cos\theta\sin\theta+\sin\theta\cos\theta & \sin^2+\cos^2\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = [1]$$

Thus, the transformation matrix for rotation in

- Anticlockwise direction will be

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

- Clockwise direction will be

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

EXAMPLE 8.1.

Write 2 × 2 transformation matrix for each of the following rotation about the origin

(*i*) counter clockwise by π

(*ii*) clock wise by $\pi/2$

Solution: (*i*) We know for counter clockwise the rotation matrix is

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

Here $\theta = \pi$

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

(*ii*) For clockwise rotation, the rotation matrix is

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

i.e.,
$$R_{\pi/2} = \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

8.3.3. Scaling

Scaling is a transformation that changes the size or shape of an object. Scaling with respect to origin can be carried out by multiplying the co-ordinate values (x, y) of each vertex of a polygon, or each endpoint of a line by scaling factors S_x and S_y respectively to produce the coordinates (x', y').

The mathematical expression for pure scaling is

$$x' = S_x * x$$
$$y' = S_y * y$$

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

or symbolically $[X'] = [T]\,[X]$

After applying scaling factor or matrix on any object, coordinate is either increased or decreased depending on the value of scaling factors S_x and S_y. If it is greater than one, then enlargement of the object will occur and if it is less than one compression will occur.

In general, for *uniform scaling* if $S_x = S_y > 1$, then a uniform compression occurs *i.e.*, the object becomes larger. If $S_x = S_y < 1$, then a uniform compression occurs *i.e.*, the object gets smaller. Non-uniform expansions or compressions occur, depending on whether S_x and S_y are individually > 1 or < 1 but unequal, such scaling is also known as *differential scaling*. While the basic object shape remains unaltered in uniform scaling, the shape and size both changes in differential scaling.

Thus, in pure uniform scaling with factors < 1 moves objects closer to the origin while factors > 1 moves object farther from origin, at the same time decreasing or increasing the object size.

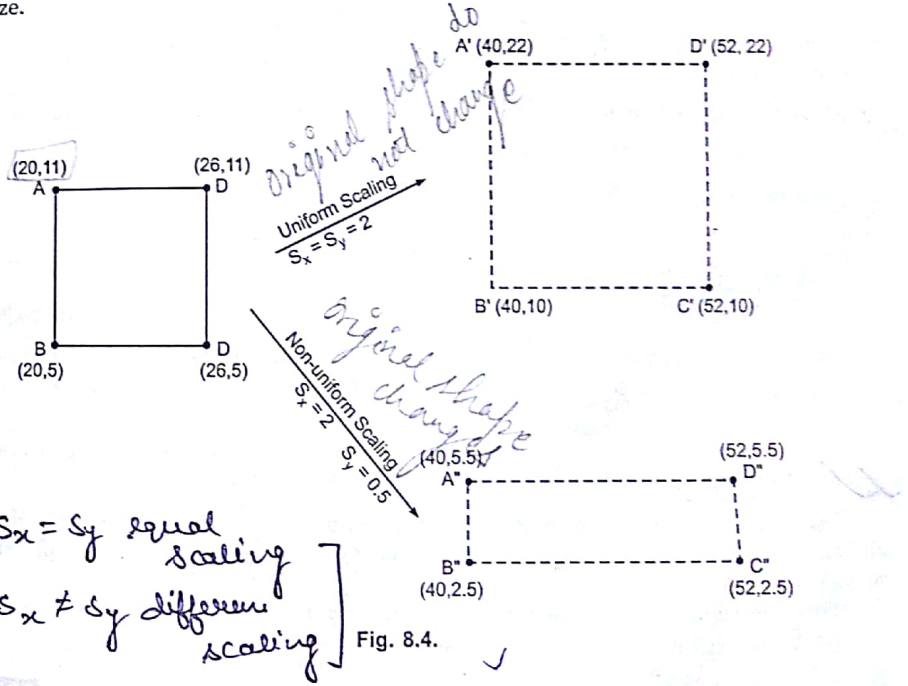


Fig. 8.4.

EXAMPLE 8.2.

Write 2 × 2 transformation matrix for each of the following scaling transformation:

(*i*) The entire picture is 3 times as large.

Scanned by CamScanner

(ii) The entire picture is 1/3 as large.
(iii) The x direction is 4 times as large and the y direction unchanged.
(iv) The y length is reduced to 2/3 their original value and the x length unchanged.
(v) The x direction reduced to 3/4 the original and y direction increased by 7/5 times.

Solution: (i) We know the scaling transformation matrix is

$$\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

Put $S_x = 3$ and $S_y = 3$

$S_{3,3} = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$ which will transform the picture three times of its original size.

(ii) Here $S_x = \frac{1}{3}$ and $S_y = \frac{1}{3}$

So $$S_{\frac{1}{3},\frac{1}{3}} = \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}$$

(iii) Here, $S_x = 4$ and $S_y = 1$ (*i.e.*, unchanged)

So, $$S_{4,1} = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$$

(iv) Here $S_x = 1$ and $S_y = \frac{2}{3}$

Therefore, $$S_{1,\frac{2}{3}} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{2}{3} \end{bmatrix}$$

(v) Here $S_x = \frac{3}{4}$ and $S_y = \frac{7}{5}$

Therefore, $$S_{\frac{3}{4},\frac{7}{5}} = \begin{bmatrix} \frac{3}{4} & 0 \\ 0 & \frac{7}{5} \end{bmatrix}$$

8.3.4. Shearing

The transformation "shearing", when applied to any object results only in distortion of shape. In shearing, the opposite and parallel layers of any object are simply slided with respect to each other.

Shearing can be done either in x-direction or y-direction.

(a) X-shear: In x-shear y co-ordinate remains unchanged, but x is changed as shown in Fig. 8.5.

Transformation matrix for x-shear is

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

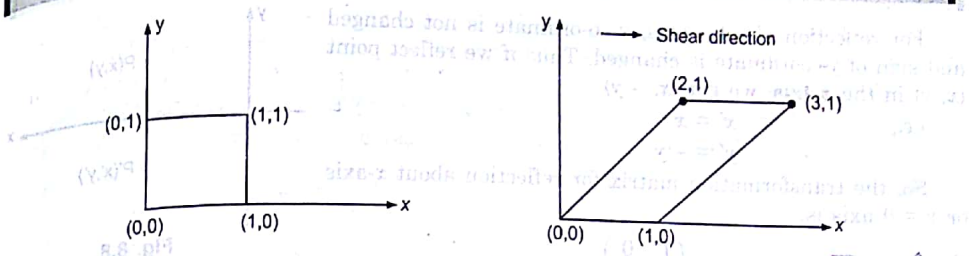


Fig. 8.5.

$$[x', y] = [x, y]^* \begin{bmatrix} 1 & 0 \\ b & 1 \end{bmatrix} = [x + yb, y]$$

y-shear: In y-shear, y co-ordinate is changed and x-remains unchanged.

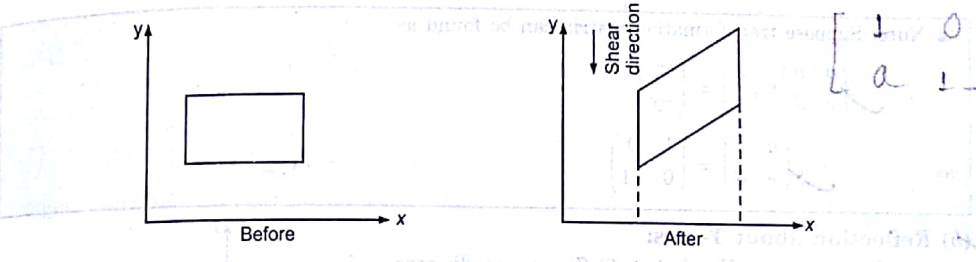


Fig. 8.6.

Transformation matrix for y-shear is

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

Thus $$[x, y'] = [x, y] * \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} = [x, xa + y]$$

8.3.5. Reflection

A reflection is a transformation that produces a mirror image of an object. In 2D reflection we consider any line in 2D plane as the mirror. The reflection is also described as the rotation by 180°.

(a) Reflection about x-axis:

(UPTU 2009)

The basic principles behind reflection transformation are:

(i) The image of an object is formed on the side opposite to where the object is lying w.r.t. mirror line.

(ii) The perpendicular distance of the object from the mirror line is same to the distance of the reflected image.

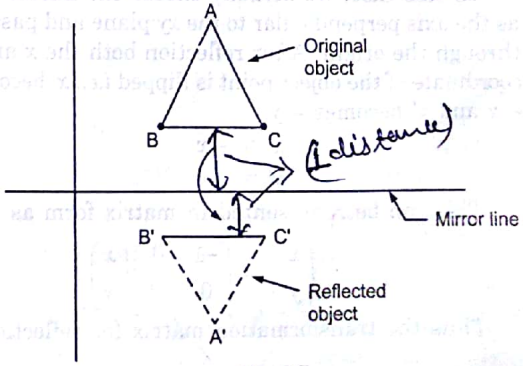


Fig. 8.7.

Scanned by CamScanner

For reflection about x-axis, x co-ordinate is not changed and sign of y-coordinate is changed. Thus if we reflect point (x, y) in the x-axis, we get $(x, -y)$

i.e., $x' = x$

$y' = -y$

So, the transformation matrix for reflection about x-axis or $y = 0$ axis is,

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

and the transformation is represented as

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

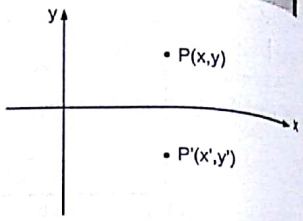


Fig. 8.8.

Note: Suppose transformation matrix can be found as

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} * \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ -y \end{pmatrix}$$

as $$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

(b) Reflection about Y-axis:

A reflection about Y-axis ($x = 0$) flips x-coordinates while y-coordinates remains the same. For reflection of $P(x, y)$ to $P'x', y)$

$x' = -x$

$y' = y$

This transformation is identified by the reflection transformation matrix as

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

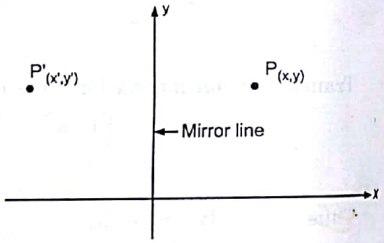


Fig. 8.9.

(c) Reflection about the Origin:

In this case, we actually choose the mirror line as the axis perpendicular to the xy plane and passing through the origin. After reflection both the x and y coordinate of the object point is flipped *i.e.*, x' becomes $-x$ and y' becomes $-y$.

i.e., $x' = -x$

$y' = -y$

This can be represented in matrix form as

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

Thus the transformation matrix for reflection about origin is

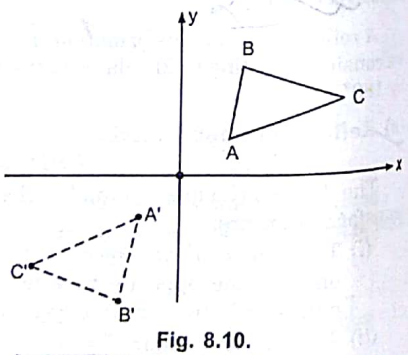


Fig. 8.10.

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

(d) Reflection about the straight line $y = x$

If we reflect (x, y) point about the line $y = x$, we get (y, x) i.e.,

$x' = y$

$y' = x$

This can be represented by,

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

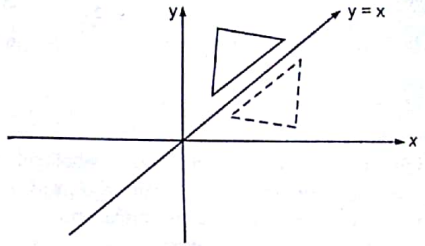


Fig. 8.11.

where $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ is the transformation matrix for reflection about line $y = -x$.

(e) Reflection about the straight line $y = -x$

If we reflect point $P(x, y)$ in the line $y = -x$ it becomes $(-y, -x)$

i.e., $x' = -y$

$y' = -x$

This can be represented by

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

where $\begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$ is the transformation matrix for reflection about line $y = -x$.

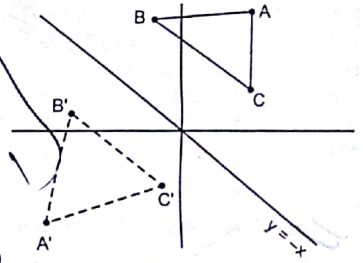


Fig. 8.12.

8.4. HOMOGENEOUS COORDINATES

(UPTU 2004)

We have seen how the shape, size, position and orientation of 2D objects. In some cases, however a desired orientation of an object may require more than one transformation to be applied successively. For example let us consider a case which requires 90° rotation of a point $\begin{pmatrix} x \\ y \end{pmatrix}$ about origin followed by reflection through the line $y = x$.

After rotation $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -y \\ x \end{pmatrix}$

This $\begin{pmatrix} x' \\ y' \end{pmatrix}$ then undergoes reflection to produce $\begin{pmatrix} x'' \\ y'' \end{pmatrix}$ as

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

These successive matrix operations can be symbolically expressed as

$[X''] = [T_M]_{y=x} [X']$

$[X''] = [T_M]_{y=x} \{[T_R]_{90°}[X]\}$

Scanned by CamScanner

i.e., [illegible] $= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

Now $[X^*] = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ -y \end{pmatrix}$

which shows the same result as before.

This process of calculating the product of matrix of a number of different transformations in a sequence is known as concatenation or combination of transformations and the resultant product matrix is referred to as *composite or concatenated transformation matrix*.

Application of concatenated transformation matrix on the object coordinates eliminates the calculation of intermediate coordinate values after each successive transformation.

But the real problem arises when there is a translation or rotation or scaling about an arbitrary point other than the origin involved among several successive transformations. The reason being the general form of expression of such transformations is not simply $[X'] = [T][X]$ involving the 2×2 array, $[T]$ containing multiplicative factors, rather it is in form $[X'] = [T_1][X] + [T_2]$ where $[T_2]$ is the additional two element column matrix containing the translational terms. Such transformations cannot be combined to form a single resultant representative matrix. This problem can be eliminated of we can combine $[T_1]$ and $[T_2]$ into a single transformation matrix. This can be done by expanding the 2×2 transformation matrix format into 3×3 form. The general 3×3 form will be something like.

$\begin{bmatrix} a & b & m \\ c & d & n \\ 0 & 0 & 1 \end{bmatrix}$ where the elements a, b, c, d of the upper left 2×2 submatrix are the multiplicative factors of $[T_1]$ and m, n are the respective x, y translational factors of $[T_2]$. But such 3×3 matrices are not comfortable for multiplication with 2×1 2D position vector matrices. Herein lies the need to include a dummy coordinate to make 2×1 position vector matrix $\begin{pmatrix} x \\ y \end{pmatrix}$ to a 3×1 matrix $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ where the third coordinate is dummy.

Now, if we multiply, $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ with a non-zero scalar 'h' then the matrix it forms is

$\begin{pmatrix} xh \\ yh \\ h \end{pmatrix}$ or symbolically say $\begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix}$ which is known as the *homogeneous coordinates* or homogeneous position vector of the same point $\begin{pmatrix} x \\ y \end{pmatrix}$ is 2D

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix} = h.\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

The extra coordinate h is known as a *weight*, which is homogeneously applied to the cartesian components. Thus a general homogeneous coordinate representation of any point $P(x, y)$ is (x_h, y_h, h) that implies $x = \frac{x_h}{h}$, $y = \frac{y_h}{h}$. As 'h' can have any non-zero value, there can be infinite number of equivalent homogeneous representation of any given point in space. e.g., (4, 6, 2), (8, 12, 4), (2, 3, 1), (–2, –3, –1) all represent the physical point (2, 3).

But so far as geometric transformation is concerned our choice is simply $h = 1$ and the corresponding homogenous coordinate triple $(x, y, 1)$ for representation of point position (x, y) in xy-plane. Other values of parameter 'h' are needed frequently in matrix formulation of three dimensional viewing transformation.

In 3×3 matrix form for homogeneous co-ordinates, the translation equations are

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

Equations of rotation and scaling with respect to coordinate origin *may* be modified as

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

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

Similarly the modified general expression for reflection may be

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

where values of a, b, c, d depend upon choice of co-ordinate axes or diagonal axis as mirror line.

8.5. COMBINED TRANSFORMATIONS *(UPTU 2009)*

To control the size of an object, we need to perform matrix operations on the position vector which defines the vertices. It is not necessary that we get the required orientation by applying single transformation, it may require more than one transformation. Since matrix multiplication is non commutative, the order of application of the transformation is important.

We can set up a matrix for any sequence of transformations as a composite transformation matrix by calculating the matrix product of the individual transformations.

(*i*) If two successive translation are applied then

$$\begin{pmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_{x1}+t_{x2} \\ 0 & 1 & t_{y1}+t_{y2} \\ 0 & 0 & 1 \end{pmatrix}$$

(*ii*) If two successive rotations θ_1 and θ_2 then

$$\begin{pmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(iii) If twio successive scaling are applied then

$$\begin{pmatrix} S_{x1}\cdot S_{x2} & 0 & 0 \\ 0 & S_{y1}\cdot S_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

8.5.1. Inverse Transformation

For each geometric transformation there exists an inverse transformation which describes just the opposite operation of that performed by the original transformation. Any transformation followed by its inverse transformation keeps an object unchanged in position, orientation, size and shape. We will use inverse transformation to nullify the effects of already applied transformation.

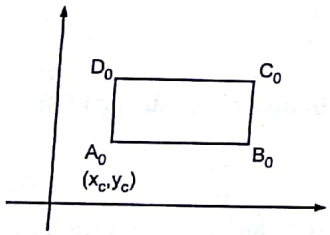
8.5.2. Rotation About an Arbitrary Point

We discussed earlier about rotation, which occurs only about the origin but now our aim is to rotate the object about a point other than origin. For this purpose homogeneous coordinate system will provide a method to accomplish rotation about any point. This can be done in the following order:

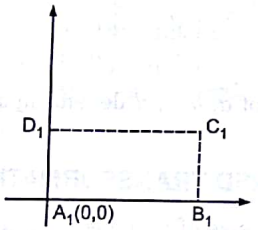
Step 1: Translate the object or body at the origin (translation)

Step 2: Rotate by any angle as given (Rotation)

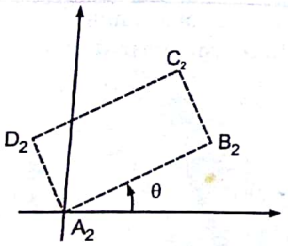
Step 3: Translate back to its original location (inverse translation).



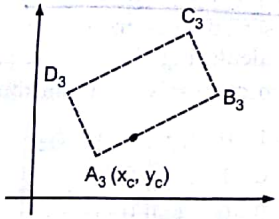
(a) Rectangle A_0 B_0 C_0 D_0 to be rotated about pivot point A_0 (x_c, y_c)



(b) Rectangle is translated by $(-x_c, y_c)$ so that A_0 becomes origin i.e., (0, 0).



(c) Rotation about origin



(d) Reverse translation and rectangle moves back to its original position

Fig. 8.13.

In matrix form, it can be shown as

$$\boxed{[T] = [T_R][R_\theta][T_R]^{-1}}$$

where $T_R \rightarrow$ Translation matrix

$R_\theta \rightarrow$ Rotation matrix by an angle $\theta°$

$[T_R]^{-1} \rightarrow$ Inverse translation matrix.

Let us obtain transformation matrix for anticlockwise rotation about point (x_c, y_c)

***(i)* Translation:** Moves (x_c, y_c) to origin $t_x = -x_c$ $t_y = -y_c$

$$T_r = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_c & -y_c & 1 \end{pmatrix}$$

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

***(iii)* Translation:** Moves the image back to its original position

$t_x = x_c$ and $t_y = y_c$

$$[T_r]^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_c & y_c & 1 \end{pmatrix}$$

Now combined transformation matrix is

$$[x' \;\; y' \;\; 1] = [x \;\; y \;\; 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_c & -y_c & 1 \end{bmatrix} \underbrace{\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_c & y_c & 1 \end{bmatrix}}$$

$$= [x \;\; y \;\; 1] \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_c & -y_c & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ x_c & y_c & 1 \end{bmatrix}}$$

$$= [x \;\; y \;\; 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_c\cos\theta + y_c\sin\theta + x_c & -x_c\sin\theta - y_c\cos\theta + y_c & 1 \end{bmatrix}$$

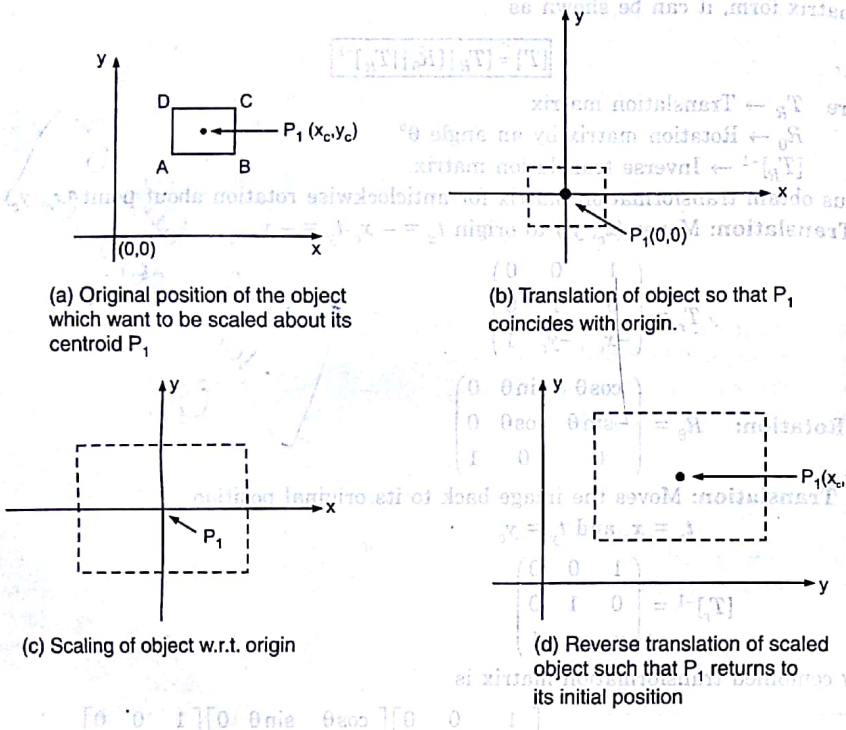
So,

$$\boxed{[x' \;\; y' \;\; 1] = [x \;\; y \;\; 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_c(\cos\theta - 1) + y_c\sin\theta & -y_c(\cos\theta - 1) - x_c\sin\theta & 1 \end{bmatrix}}$$

8.5.3. General Fixed-Point Scaling

The scaling matrix discussed previously performs scaling about the origin. But sometimes it may be required to perform scaling without altering the objects position.

Scanned by CamScanner



(a) Original position of the object which want to be scaled about its centroid P_1

(b) Translation of object so that P_1 coincides with origin.

(c) Scaling of object w.r.t. origin

(d) Reverse translation of scaled object such that P_1 returns to its initial position

Fig. 8.14.

Thus, following three steps will have to be followed

1. Translate point to origin
2. Perform scaling
3. Inverse translation

Hence,
$$[T] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_c & -y_c & 1 \end{bmatrix} \underbrace{\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_c & y_c & 1 \end{bmatrix}}$$

$$= \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_c & -y_c & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ x_c & y_c & 1 \end{bmatrix}} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ x_c(1-S_x) & y_c(1-S_y) & 1 \end{bmatrix}$$

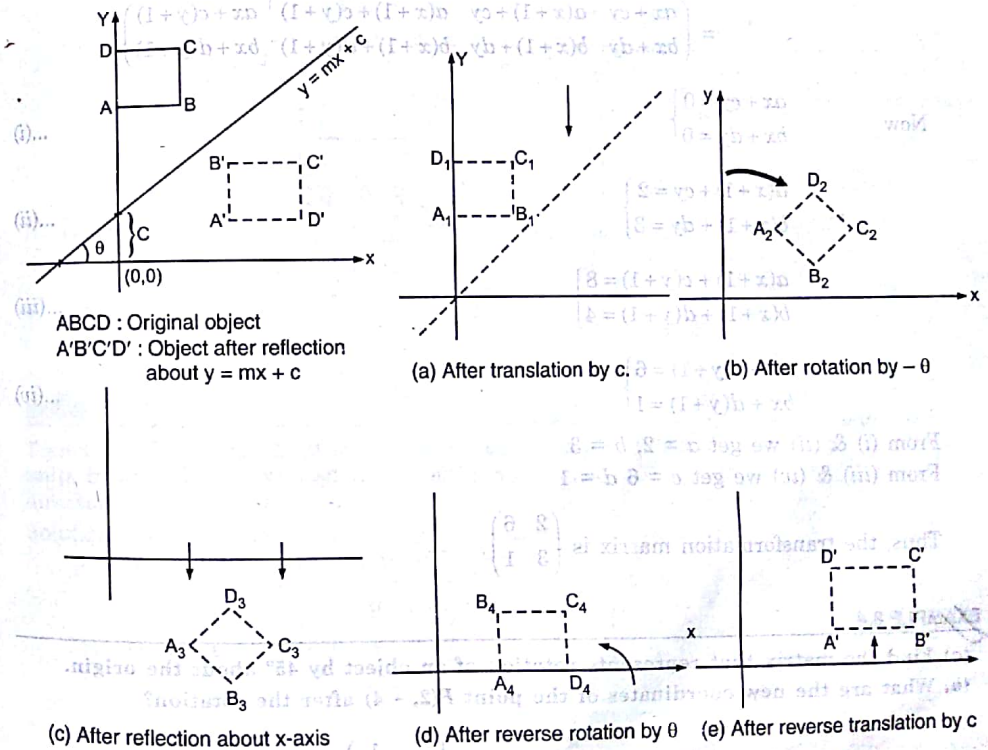
8.5.4. Reflection Through an Arbitrary Line

We have already discussed reflection through the line $x = 0$, $y = 0$, $y = x$, $y = -x$. All these lines pass through origin. But when reflection is to be performed about a line that neither passes through the origin nor is parallel to the co-ordinate axis can be solved using the following steps:

Step-1: Translate the line and the object so that the line passes through the origin.
Step-2: Rotate the line and the object about the origin until the line is coincident with one of the coordinate axis about which we are familiar to perform reflection.
Step-3: Refect the object through the co-ordinate axis.
Step-4: Apply the inverse rotation about the origin to shift the line at translated position.
Step-5: Apply inverse translation to send back the object (i.e., line) to its original position.
In matrix rotation

$$\boxed{[T] = [T_R][R_\theta][R_{ref}][R_\theta]^{-1}[T_R]^{-1}}$$

where $T_R \rightarrow$ Translation matrix
$R_\theta \rightarrow$ Matrix for rotation by angle θ
$R_{ref} \rightarrow$ Reflection about any axis
$[R_\theta]^{-1} \rightarrow$ Inverse rotation
$[T_R]^{-1} \rightarrow$ Inverse translation



ABCD : Original object
A'B'C'D' : Object after reflection about y = mx + c

(a) After translation by c.

(b) After rotation by $-\theta$

(c) After reflection about x-axis

(d) After reverse rotation by θ

(e) After reverse translation by c

Fig. 8.15.

EXAMPLE 8.3.

A unit square is transformed by a 2 × 2 transformation matrix. The resulting position vectors are

Scanned by CamScanner

$$\begin{pmatrix} 0 & 2 & 8 & 6 \\ 0 & 3 & 4 & 1 \end{pmatrix}$$

what is the transformation matrix?

Solution: Suppose the unit square have coordinates

(x, y)
$(x + 1, y)$
$(x + 1, y + 1)$
$(x, y + 1)$

and let the transformation matrix be $\begin{pmatrix} a & c \\ b & d \end{pmatrix}$

So, $\begin{pmatrix} 0 & 2 & 8 & 6 \\ 0 & 3 & 4 & 1 \end{pmatrix} = \begin{pmatrix} a & c \\ b & d \end{pmatrix}\begin{pmatrix} x & x+1 & x+1 & x \\ y & y & y+1 & y+1 \end{pmatrix}$

$$= \begin{pmatrix} ax+cy & a(x+1)+cy & a(x+1)+c(y+1) & ax+c(y+1) \\ bx+dy & b(x+1)+dy & b(x+1)+d(y+1) & bx+d(y+1) \end{pmatrix}$$

Now
$$\left.\begin{aligned} ax+cy &= 0 \\ bx+dy &= 0 \end{aligned}\right\} \quad ...(i)$$

$$\left.\begin{aligned} a(x+1)+cy &= 2 \\ b(x+1)+dy &= 3 \end{aligned}\right\} \quad ...(ii)$$

$$\left.\begin{aligned} a(x+1)+c(y+1) &= 8 \\ b(x+1)+d(y+1) &= 4 \end{aligned}\right\} \quad ...(iii)$$

$$\left.\begin{aligned} ax+c(y+1) &= 6 \\ bx+d(y+1) &= 1 \end{aligned}\right\} \quad ...(iv)$$

From (*i*) & (*ii*) we get $a = 2$, $b = 3$.
From (*iii*) & (*iv*) we get $c = 6$ $d = 1$

Thus, the transformation matrix is $\begin{pmatrix} 2 & 6 \\ 3 & 1 \end{pmatrix}$.

EXAMPLE 8.4.

(*a*) Find the matrix that represents rotation of an object by 45° about the origin.
(*b*) What are the new coordinates of the point $P(2, -4)$ after the rotation?

Solution: (*a*) $R_{45°} = \begin{pmatrix} \cos 45° & -\sin 45° \\ \sin 45° & \cos 45° \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$

(*b*) The new co-ordinates can be found by

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}\begin{pmatrix} 2 \\ -4 \end{pmatrix} = \begin{pmatrix} \sqrt{2}+2\sqrt{2} \\ \sqrt{2}-2\sqrt{2} \end{pmatrix}$$

EXAMPLE 8.5.

A triangle is defined by $\begin{pmatrix} 2 & 4 & 4 \\ 2 & 2 & 4 \end{pmatrix}$

Find the transformed coordinates after the following transformation
(*i*) 90° rotation about origin
(*ii*) reflection about line $y = -x$.

Solution: (*i*) After 90° rotation about origin, the transformed coordinate are

$$\begin{pmatrix} \cos 90° & -\sin 90° & 0 \\ \sin 90° & \cos 90° & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 2 & 4 & 4 \\ 2 & 2 & 4 \\ 1 & 1 & 1 \end{pmatrix}$$

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

Finally after reflection about line $y = -x$, the transformation matrix is

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

EXAMPLE 8.6.

Translate the square *ABCD* whose co-ordinate are *A*(0, 0), *B*(3, 0), *C*(3, 3), and *D*(0, 3) by 2 units in both directions and then scale it by 1.5 units in *x*-direction and 0.5 units in *y*-direction.

Solution: Here first operation is translation and second operation is scaling.

$T_x = 2$ and $T_y = 2$

Translation matrix $= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix}$

Square *ABCD* in matrix form $= \begin{bmatrix} 0 & 0 & 1 \\ 3 & 0 & 1 \\ 3 & 3 & 1 \\ 0 & 3 & 1 \end{bmatrix}$

Performing translation operation, we get

Scanned by CamScanner

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

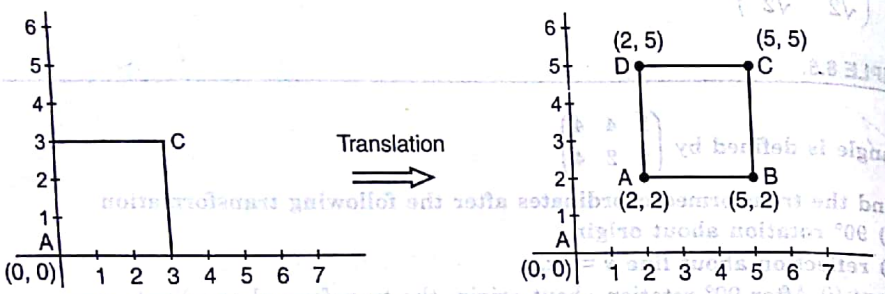


Fig. 8.16.

Now the 2nd operation is scaling so $S_x = 1.5$ $S_y = 0.5$

and scaling matrix $= \begin{bmatrix}S_x&0&0\\0&S_y&0\\0&0&1\end{bmatrix} = \begin{bmatrix}1.5&0&0\\0&0.5&0\\0&0&1\end{bmatrix}$

After scaling we will get

$$\begin{bmatrix}2&2&1\\5&2&1\\5&5&1\\2&5&1\end{bmatrix}\begin{bmatrix}1.5&0&0\\0&0.5&0\\0&0&1\end{bmatrix}=\begin{bmatrix}3&1&1\\7.5&1&1\\7.5&2.5&1\\3&2.5&1\end{bmatrix}$$

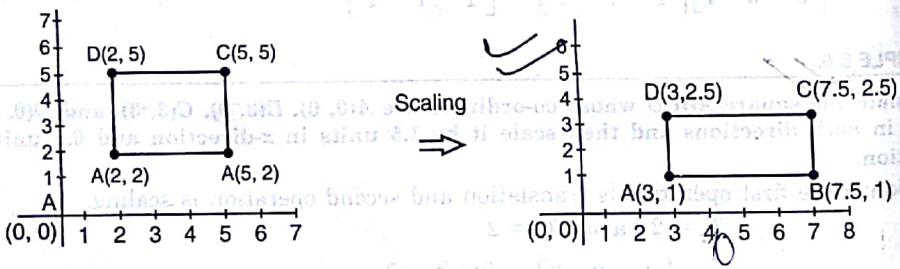


Fig. 8.17.

After transformation, new co-ordinates are

$A(3,1)$
$B(7.5, 1)$
$C(7.5, 2.5)$
$D(3, 2.5)$

EXAMPLE 8.7.

Perform a 45° rotation of triangle $A(0, 0)$, $B(1, 1)$, $C(5, 2)$ (UPTU 2003)
(a) about the origin and (b) about point $P(-1, -1)$.

Solution: Triangle ABC in matrix form is given as

$$\begin{bmatrix}0&0&1\\1&1&1\\5&2&1\end{bmatrix} \quad \text{or} \quad \begin{bmatrix}0&1&5\\0&1&2\\1&1&1\end{bmatrix}.$$

(a) Matrix of rotation is

$$R_{45°} = \begin{bmatrix}\cos 45° & -\sin 45° & 0\\ \sin 45° & \cos 45° & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}\frac{\sqrt{2}}{2} & \frac{-\sqrt{2}}{2} & 0\\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0\\ 0 & 0 & 1\end{bmatrix}$$

Now find the co-ordinates A' B' C' of the rotated triangle ABC can be found as

$$[A'\ B'\ C'] = R_{45°}\cdot\begin{bmatrix}0&1&5\\0&1&2\\1&1&1\end{bmatrix} = \begin{bmatrix}\frac{\sqrt{2}}{2} & \frac{-\sqrt{2}}{2} & 0\\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}0&1&5\\0&1&2\\1&1&1\end{bmatrix} = \begin{bmatrix}0 & 0 & \frac{3\sqrt{2}}{2}\\ 0 & \sqrt{2} & \frac{7\sqrt{2}}{2}\\ 1 & 1 & 1\end{bmatrix}$$

Thus $A' = (0, 0)$

$B' = (0, \sqrt{2})$

$C' = \left(\frac{3\sqrt{2}}{2}, \frac{7\sqrt{2}}{2}\right)$

(b) The rotation matrix is

$$\begin{bmatrix}1&0&-1\\0&1&-1\\0&0&1\end{bmatrix}\begin{bmatrix}\frac{\sqrt{2}}{2} & \frac{-\sqrt{2}}{2} & 0\\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1&0&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & -1\\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & (\sqrt{2}-1)\\ 0 & 0 & 1\end{bmatrix}$$

Now, $[A'\ B'\ C'] = \begin{bmatrix}\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & -1\\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & (\sqrt{2}-1)\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}0&1&5\\0&1&2\\1&1&1\end{bmatrix}$

Scanned by CamScanner

$$= \begin{pmatrix} -1 & -1 & \left(\frac{3}{2}\sqrt{2}-1\right) \\ \left(\sqrt{2}-1\right) & \left(2\sqrt{2}-1\right) & \left(\frac{9}{2}\sqrt{2}-1\right) \\ 1 & 1 & 1 \end{pmatrix}$$

Thus $A' = \left(1-, \sqrt{2}-1\right)$

$B' = \left(1-, 2\sqrt{2}-1\right)$

$C' = \left(\frac{3}{2}\sqrt{2}-1, \frac{9}{2}\sqrt{2}-1\right)$

EXAMPLE 8.8.

Find the transformation matrix that transforms the square *ABCD* whose center is at (2, 2) is reduced to half of its size, with center still remaining at (2, 2). The coordinates of square *ABCD* are *A*(0, 0) *B*(0, 4) *C*(4, 4) and *D*(4, 0). Find the co-ordinates of new square.

Solution: Square ABCD in matrix form as

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 4 & 1 \\ 4 & 4 & 1 \\ 4 & 0 & 1 \end{pmatrix}$$

For reducing square *ABCD* to half of its size we scale it by $S_x = \frac{1}{2}$ and $S_y = \frac{1}{2}$. So, scaling matrix is

$$\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

After scaling we get

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 4 & 1 \\ 4 & 4 & 1 \\ 4 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 2 & 1 \\ 2 & 2 & 1 \\ 2 & 0 & 1 \end{pmatrix}$$

After scaling the co-ordinates of square *ABCD* are

$A = (0, 0)$

$B = (0, 2)$ and now the centre is (1, 1) but we want the centre of *ABCD* (2, 2)

Therefore,

$C = (2, 2)$ we translate the square *ABCD* by $T_x = 1$ and $T_y = 1$.

$D = (2, 0)$

Translation matrix

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

After translation we get

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

Thus, the new co-ordinates for square *ABCD* are

$A \rightarrow (1, 1)$

$B \rightarrow (1, 3)$

$C \rightarrow (3, 3)$

$D \rightarrow (3, 1)$

EXAMPLE 8.9.

Consider the square A(1, 0) *B*(0, 0) *C*(0, 1) *D*(1, 1). Rotate the square ABCD by 45° clockwise about A(1, 0).

Solution: Square *ABCD* in matrix form as follows.

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

Matrix for rotation in clockwise about origin.

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

We have to rotate square *ABCD* about point *A*(1, 0). We first translate square *ABCD* by $T_x = -1$ and $T_y = 0$ *i.e.*,

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Now we rotate in clockwise $\theta = 45°$

$$\begin{pmatrix} \cos 45° & -\sin 45° & 0 \\ \sin 45° & \cos 45° & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 1 \\ 0 & \sqrt{2} & 1 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 1 \end{pmatrix}$$

Scanned by CamScanner

Now, we translate square ABCD back to its position $T_x = 1$ & $T_y = 0$ *i.e.*,

$$\begin{pmatrix} 0 & 0 & 1 \\ -1/\sqrt{2} & 1/\sqrt{2} & 1 \\ 0 & \sqrt{2} & 1 \\ 1/\sqrt{2} & 1/\sqrt{2} & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 \\ 1-1/\sqrt{2} & 1/\sqrt{2} & 1 \\ 1 & \sqrt{2} & 1 \\ 1+1/\sqrt{2} & 1/\sqrt{2} & 1 \end{pmatrix}$$

EXAMPLE 8.10.

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

Solution: First we write the required transformation matrix.

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

Triangle ABC in matrix form as follows

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

Now, the new transformation matrix

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

i.e., the new co-ordinates after transformation are

$A \rightarrow (-5, -2)$
$B \rightarrow (-3, 0)$
$C \rightarrow (5, 2)$

EXAMPLE 8.11.

Prove that two-dimensional rotation and scaling commutative if $S_x = S_y$ or $\theta = n\pi$.

Solution: The transformation matrix for rotation about origin in anticlockwise direction.

$$\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$ and scaling matrix is $$\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

To prove commutative property holds for $S_x = S_y$

R.S = S.R

i.e., $$\text{R.S} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} S_x\cos\theta & S_x\sin\theta & 0 \\ -S_x\sin\theta & S_x\cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\text{S.R.} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} S_x\cos\theta & S_x\sin\theta & 0 \\ -S_x\sin\theta & S_x\cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Thus R.S = S.R

Now for $\theta = n\pi$

$$\text{R.S} = \begin{pmatrix} \cos n\pi & \sin n\pi & 0 \\ -\sin n\pi & \cos n\pi & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\text{S.R} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \cos n\pi & \sin n\pi & 0 \\ -\sin n\pi & \cos n\pi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Thus RS = SR if $S_x = S_y$ or $\theta = n\pi$

EXAMPLE 8.12.

The reflection along the line $y = x$ is equivalent to the reflection along the x-axis followed by counter clockwise rotation by θ degrees. Find the value of θ.

Solution: The transformation matrix for reflection about the line $y = x$ is

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \ldots(i)$$

Reflection about x-axis, the matrix is $\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and for counter clockwise rotation by θ about origin is

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

For successive application, the resultant transformation is

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

It is given that

Scanned by CamScanner

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

i.e., $\cos\theta = 0$ and $\sin\theta = 1$ and $-\cos\theta = 0 \Rightarrow \theta = 90°$

EXAMPLE 8.13.

Show that the 2 × 2 matrix

$$[T] = \begin{bmatrix}\frac{1-t^2}{1+t^2} & \frac{2t}{1+t^2}\\ \frac{-2t}{1+t^2} & \frac{1-t^2}{1+t^2}\end{bmatrix}$$ **represents pure rotation.**

Solution: We know that for pure rotational transformation determinant of the transformation matrix is always equal to 1.

So, the determinant of $[T] = \left\{\left(\frac{1-t^2}{1+t^2}\right)^2 - \left(-\frac{2t}{1+t^2}\right)^2\right\} = \frac{(1-t^2)^2}{(1+t^2)^2} + \frac{4t^2}{(1+t^2)^2} = \frac{(1-t^2)^2+4t^2}{(1+t^2)^2}$

$$= \frac{1-2t^2+t^4+4t^2}{(1+t^2)^2} = \frac{(1+t^2)^2}{(1+t^2)^2} = 1$$

EXAMPLE 8.14.

Show that a 2D reflection through *X* axis followed by a 2D reflection through the line $y = -x$ is equivalent to pure rotation about the origin.

Solution: The transformation matrix for reflection about x-axis is $\begin{pmatrix}1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{pmatrix}$ and reflection through $y = -x$ is $\begin{pmatrix}0 & -1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}$

When applied successively, we get

$$\begin{pmatrix}1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}0 & -1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix} = \begin{pmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}$$

The transformation matrix for rotation about origin by an angle $\theta = 270°$ is

$$\begin{pmatrix}\cos 270° & -\sin 270° & 0\\ \sin 270° & \cos 270° & 0\\ 0 & 0 & 1\end{pmatrix} = \begin{pmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}$$

Note: In general, if two pure reflection transformation about line passing through the origin are plied successively, the result is a pure rotation about origin.

EXAMPLE 8.15.

Show that transformation matrix for a reflection about $y = -x$ is equivalent to reflection relative to the y axis followed by a counter clockwise rotation by 90°.

Solution: Transformation matrix for reflection about line $y = -x$ is $\begin{pmatrix}0 & -1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}$

Transformation matrix for reflection relative to y axis is $\begin{pmatrix}-1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}$

and the transformation matrix for counterclockwise rotation is

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

Here, $\theta = 90°$

So, $\begin{pmatrix}\cos 90° & \sin 90° & 0\\ -\sin 90° & \cos 90° & 0\\ 0 & 0 & 1\end{pmatrix} = \begin{pmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}$

When applied successively, we get

$$\begin{pmatrix}-1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix} = \begin{pmatrix}0 & -1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}$$

which is equal to the transformation matrix for reflection about line $y = -x$.

EXAMPLE 8.16.

A mirror is placed vertically such that it passes through the points (10, 0) and (0, 10). Find the reflected view of triangle *ABC* with coordinates *A*(5, 50), *B*(20, 40) *C*(10, 70).

Solution: We plot the mirror passing through (0, 10) and (10, 0)

From figure we easily get $\tan\theta = \frac{10}{10} = 1$, which implies that $\theta = 45°$

To make the line coincident with the x-axis we first translate it to make it pass through origin and then rotate it by $\theta = 45°$ about origin.

Co-ordinates of triangle ABC in matrix form is

$$\begin{pmatrix}5 & 50 & 1\\ 20 & 40 & 1\\ 10 & 70 & 1\end{pmatrix}$$

We translate mirror, so that it passes through origin.

$t_x = 0 \quad t_y = -10$

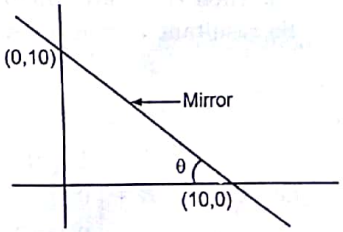


Fig. 8.18.

Scanned by CamScanner

$$T_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -10 & 1 \end{pmatrix}$$

Immediately we write inverse transformation matrix for translation by $t_x = 0$ and $t_y = 10$ is

$$T_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 10 & 1 \end{bmatrix}$$

Now we rotate the mirror by 45° anticlockwise so that it matches with origin

$$R_1 = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos 45° & \sin 45° & 0 \\ -\sin 45° & \cos 45° & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Immediately, we write inverse transformation matrix for rotation by 45°

$$R_1^{-1} = \begin{pmatrix} \cos(-45°) & \sin(-45°) & 0 \\ -\sin(-45°) & \cos(-45°) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Transformation matrix for reflection about x-axis is

$$R_{ref} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The steps are:

1. Translate the mirror and object so that it passes through origin *i.e.*, T_1
2. Rotate mirror and object by 45° in anticlockwise *i.e.*, R_1
3. Now mirror matches with x-axis then reflect triangle ABC about x-axis *i.e.*, R_{ref}
4. Rotate mirror and object by 45° clockwise *i.e.*, R_1^{-1}
5. Then translate mirror and object back to its position matrix *i.e.*, T_1^{-1}.

So resultant transformation matrix is

$$R = T_1 * R_1 * R_{ref} * R_1^{-1} * T_1^{-1}$$

i.e.,

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -10 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 10 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{10}{\sqrt{2}} & -\frac{10}{\sqrt{2}} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 10 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{10}{\sqrt{2}} & \frac{10}{\sqrt{2}} & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 10 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 10 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 10 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 10 & 10 & 1 \end{pmatrix}$$

Now, the new co-ordinates for Δ ABC

$$\begin{pmatrix} 5 & 50 & 1 \\ 20 & 40 & 1 \\ 10 & 70 & 1 \end{pmatrix} \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 10 & 10 & 1 \end{pmatrix} = \begin{pmatrix} -40 & 5 & 1 \\ -30 & -10 & 1 \\ -60 & 0 & 1 \end{pmatrix}$$

Thus after reflection, the new co-ordinates are

$A \rightarrow (-40, 5)$
$B \rightarrow (-30, -10)$
$C \rightarrow (-60, 0)$

EXAMPLE 8.17.

Prove that simultaneous shearing in both directions (x & y directions) is not equal to the composition of pure shear along x-axis followed by pure shear along y-axis.

Solution: We know the simultaneous shearing

$$S_h = \begin{pmatrix} 1 & a \\ b & 1 \end{pmatrix} \quad \ldots(i)$$

Shearing in x-direction is $\begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix}$ and in y-direction is $\begin{pmatrix} 1 & 0 \\ b & 1 \end{pmatrix}$. Therefore, shearing in x-direction followed by y-direction is

$$\begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ b & 1 \end{pmatrix} = \begin{pmatrix} 1+ab & a \\ b & 1 \end{pmatrix}$$

which is not equal to Eq^n(i).

Scanned by CamScanner

9 CHAPTER Nine

WINDOWS AND CLIPPING

Chapter Outline

- Introduction
- Window-to View Port Co-Ordinate Transformation
- Clipping
  - Point Clipping
  - Line Clipping
  - Polygon Clipping
  - Curve Clipping
  - Text Clipping
  - Interior and Exterior Clipping
  - Generalized Clipping
- Multiple Windowing
- The Viewing TransformatioN

9.1. INTRODUCTION

When drawing are too complex, they become difficult to read. In such situations, it is useful to display only those portions of the drawing that are of immediate interest. This gives the effect of looking at the image through a window. Furthermore it is desirable to enlarge these portions to take full advantage of the display surface. The method for selecting, and enlarging portions of a drawing is called **Windowing.** The technique for not showing that part of the drawing which one is not interested in is called **'clipping'.**

9.2. THE VIEWING TRANSFORMATION

Displaying an image of a picture involves mapping the co-ordinates of the points and lines that form the picture into the appropriate co-ordinates on the device or workstation where the image is to be displayed. This is done through the use of co-ordinate transformations known as *viewing transformations.* To perform a viewing transformation, we deal with a finite region in the WCS (World Co-ordinate System) called a *Window.* The window can be mapped directly on the display area of the device or on to a subregion of the display called a **'viewport'.** Thus, a world-coordinate area selected for display is called a window and an area on a display device to which a window is mapped is called a viewport, *i.e.* the window defines what is to be viewed the viewport defines where it is to be displayed.

If we are changing the position of window by keeping the viewpart location constant, then the different part of the object is displayed at the same position on the display device. Similarly if we change the location of viewport then we will see the same part of the object drawn at different places on screen.

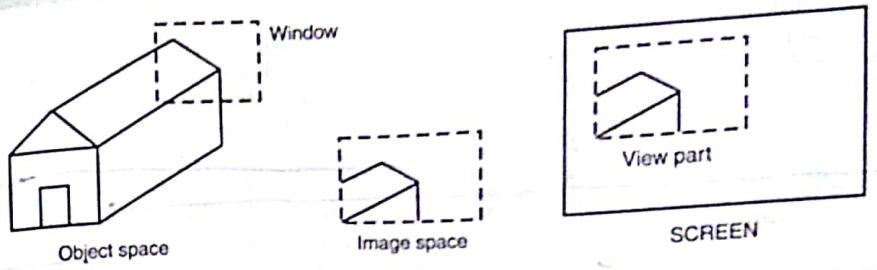


Fig. 9.1(a)

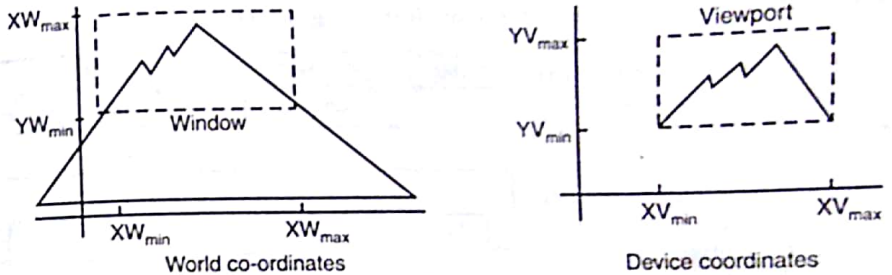
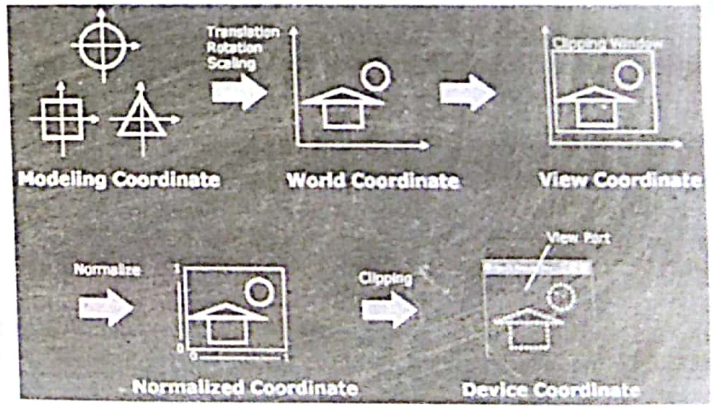
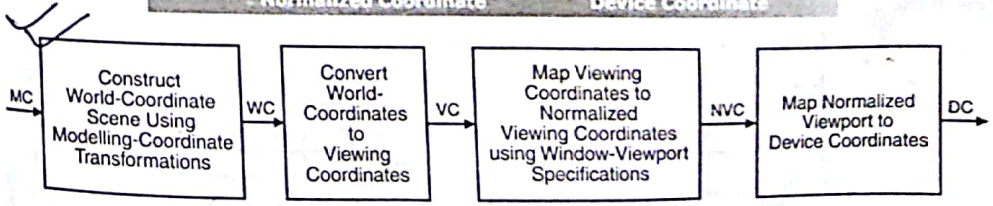


Fig. 9.1(b)

Viewing Pipeline

Some graphics packages that provide window and viewport operations allow only standard rectangles, but a more general approach is to allow the rectangular window to have any orientation. In this case, we carry out the viewing transformation in several steps, as indicated below.





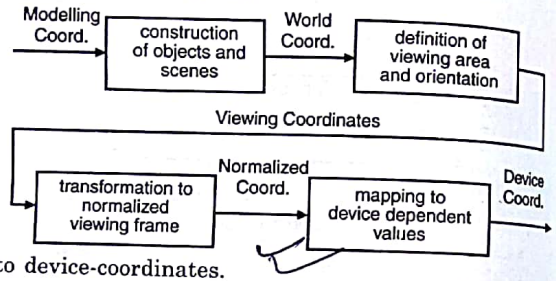
Scanned by CamScanner

First, we construct the scene in world coordinates using the output primitives. Next to obtain a particular orientation for the window, we can set up a two-dimensional viewing-coordinate system in the world-coordinate plane, and define a window in the viewing-coordinate system. The viewing coordinate reference frame is used to provide a method for setting up arbitrary orientations for rectangular windows. Once the viewing reference frame is established we can transform descriptions in world coordinates to viewing coordinates.

We then define a viewport in normalized coordinates (in the range from 0 to 1) and map the viewing-coordinate description of the scene to normalized coordinates. At the final step, all parts of the picture that outside the viewport are clipped, and the contents of the viewport are transferred to device coordinates.

2D viewing pipeline can be achieved by following steps

- Construct world-coordinate scene using modeling-coordinate transformations.
- Convert world-coordinates to viewing coordinates.
- Transform viewing-coordinates to normalized-coordinates (ex: between 0 and 1, or between – 1 and 1).
- Map normalized-coordinates to device-coordinates.



9.3. WINDOW-TO VIEW PORT CO-ORDINATE TRANSFORMATION *(UPTU 2009)*

In general the mapping of a part of world co-ordinate scene to device co-ordinates is referred as *viewing transformation*. Sometimes it is also called *Window-viewport transformation* or *windowing transformation*.

The viewing transformation is a process of 3 steps.

Step 1: The object together with its window is translated until the lower-left corner of the window is at the origin.

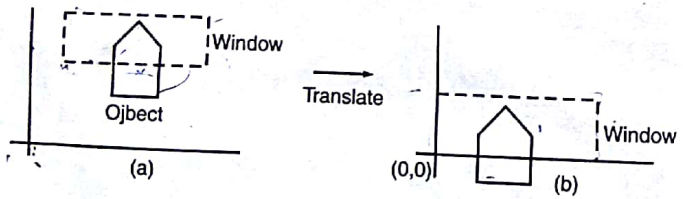


Fig. 9.2.

Step 2: The object and the window are now scaled until the window has the dimension same as view port. In other words we are converting the object into image and window in view port.

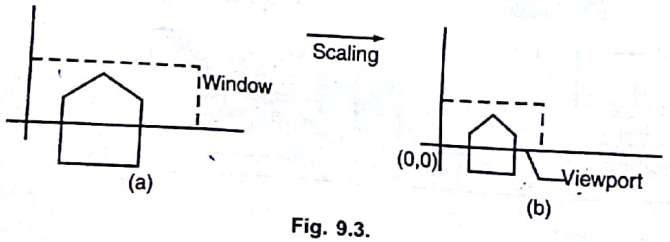


Fig. 9.3.

Step 3: The final transformation step is another translation to move the viewport to its correct position on the screen.

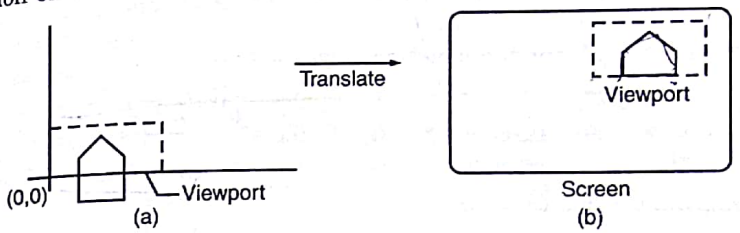


Fig. 9.4.

So, the viewing transformation performs three steps:

1. Translation
2. Scaling
3. Translation.

EXAMPLE 9.1

Suppose there is a rectangle ABCD whose co-ordinates are A(1, 1), B(4, 1), C(4, 4), D(1, 4) and the window co-ordinates are (2, 2), (5, 2), (5, 5), (2, 5) and the given viewport location is (0.5, 0), (1, 0), (1, 0.5), (0.5, 0.5). Calculate the viewing transformation matrix.

Solution: First we draw the rectangle *ABCD* and window

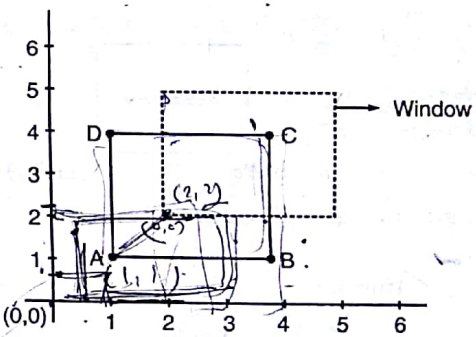


Fig. 9.5.

For viewing transformation, first step is translation. We have to shift left lower corner of window to origin, *i.e.*

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -2 & 1 \end{vmatrix}$$

Now the second step is scaling. Here we have to scale window to view port *i.e.* old dimensions will be of windows dimensions and new will be of viewport.

Scaling in x-direction *i.e.*

$$S_x = \frac{\text{width of viewport}}{\text{width of window}} = \frac{1-0.5}{5-2} = \frac{0.5}{3} = 0.16$$

Similarly $$S_y = \frac{\text{heightof viewport}}{\text{height of window}} = \frac{1-0.5}{5-2} = \frac{0.5}{3} = 0.16$$

Thus the scaling matrix will be $\begin{vmatrix} 0.16 & 0 & 0 \\ 0 & 0.16 & 0 \\ 0 & 0 & 1 \end{vmatrix}$

In third step, we have to translate the viewport to the desired location on the screen. So translation matrix will be

Scanned by CamScanner

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0 & 1 \end{vmatrix}$$

Hence viewing transformation matrix will be

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -2 & 1 \end{vmatrix} \times \begin{vmatrix} 0.16 & 0 & 0 \\ 0 & 0.16 & 0 \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0.16 & 0 & 0 \\ 0 & 0.16 & 0 \\ 0.18 & -0.32 & 1 \end{vmatrix}$$

In general, the viewing transformation is

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -W_{XL} & -W_{YL} & 1 \end{vmatrix} = \begin{vmatrix} \frac{(V_{XH}-V_{XL})}{(W_{XH}-W_{XL})} & 0 & 0 \\ 0 & \frac{(V_{YH}-V_{YL})}{(W_{YH}-W_{YL})} & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ V_{XL} & V_{YL} & 1 \end{vmatrix}$$

The variables have been named according to the rule that V stands for viewport W for windows. X for the position of a vertical boundary and Y for a horizontal boundary. H for the high boundary and L for the low boundary, *i.e.*

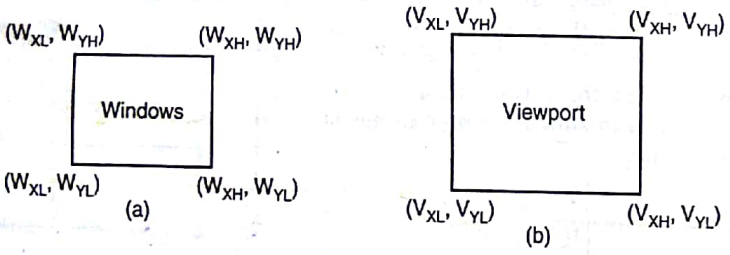


Fig. 9.6.

After multiplication the generalized viewing transformation matrix will be

$$\begin{vmatrix} \frac{V_{XH}-V_{XL}}{W_{XH}-W_{XL}} & 0 & 0 \\ 0 & \frac{V_{YH}-V_{YL}}{W_{YH}-W_{YL}} & 0 \\ (V_{XL}-W_{XL}).\frac{(V_{XH}-V_{XL})}{(W_{XH}-W_{XL})} & (V_{YL}-W_{YL}).\frac{V_{YH}-V_{YL}}{W_{YH}-W_{YL}} & 1 \end{vmatrix}$$

If the height and width of the window do not have the same properties as the height and width of the veiwport, then the viewing transformation will cause some distortion of the image.

9.4. CLIPPING

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 clipped is called a **clip window.**

Applications of clipping include extracting part of a defined scene for viewing, identifying visible surfaces in three-dimensional views; antialiasing line segments or object boundaries; creating objects using solid-modelling procedures, displaying a multiwindow environment, and drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing or duplicating. Depending on the application, the clip window can be a general polygon or it can even have curved boundaries. Clipping algorithms can be applied in world coordinats, so that only the contents of the window interior are mapped to device coordinates. Alternately the complete world coordinate picture can be mapped first to device coordinates or normalized device coordinates, then clipped against the viewport boundaries. World-coordinate clipping removes those primitives outside the window from further consideration, thus eliminating the processing necessary to transform those primitives to device space. Viewport clipping, on the other hand, can reduce calculations by allowing concatenation of viewing and geometric transformation matrices. But viewport clipping does require that the transformation to device coordinates be performed for all objects, including those outside the window area.

In the following sections, we consider algorithms for clipping the following primitive types:

1. Point clipping
2. Line clipping
3. Polygon clipping (Area)
4. Curve clipping
5. Text clipping.

9.5. POINT CLIPPING

Assuming that the clip window is a rectangle in standard position, we save a point $P(x, y)$ for display if the following inequalities are satisfied.

$$\left.\begin{aligned} x\omega_{min} \le x \le x\omega_{max} \\ y\omega_{min} \le y \le y\omega_{max} \end{aligned}\right\}$$ (world boundaries or viewport boundaries)

where the edges of the clip window $(x\omega_{min}, x\omega_{max}, y\omega_{min}, y\omega_{max})$ can be either the world-coordinate window boundaries or viewport boundaries.

If any one of these four inequalities is not satisfied, the point is clipped *i.e.*, not saved for display.

9.6. LINE CLIPPING

Lines that do not intersect the clipping window are either completely inside the window or completely outside the window. All line segments fall into one of the following clipping categories:

1. **Visible:** Both end points of the line segment lie within the window.
2. **Non-visible:** When line definitely lies outside the window. This will occur if the line segment from (x_1, y_1) to (x_2, y_2) satisfies any one of the following four inequalities:

$$\begin{array}{|ll|} \hline x_1, x_2 > x_{max} & y_1, y_2 > y_{max} \\ x_1, x_2 < x_{min} & y_1, y_2 < y_{min} \\ \hline \end{array}$$

3. **Partially visible (or clipping condidate):** A line is partially visible when a part of its lies within the window.

Line clipping algorithms include the *Cohen-Sutherland method*, the *Liang-Barsky method*, and the *Nicholl-Lee-Nicholl method*. The Cohen-Sutherland method is widely used. In this region codes are used to identify the position of line endpoints relative to the rectangular,

Scanned by CamScanner

clipping window boundaries. Lines that cannot be immediately identified as completely inside the window or completely outside are then clipped against window boundaries. *Liang and Barsky* use a parametric line representation to set up a more efficient line-clipping procedure that reduces intersection calculations. The *Nicholl-Lee-Nicholl method* uses more region testing in the *xy* plane to reduce intersection calculations even further.

The basic principles (assuming the clip window *W* is convex) are:

- If both endpoints of a line segment fall within *W*, then display the line segment.
- If both endpoints of a line segment fall outside of *W* because they violate the same point clipping inequality, then do not display the line segment.
- If one endpoint falls within *W* and another falls outside, then part of the line segment is displayed.
- If both endpoints fall outside *W*, but do not violate a common point clipping inequality, then part of the line may be visible.

9.6.1 Cohen-Sutherland Algorithm

In computer graphics, the **Cohen-Sutherland** algorithm is a line clipping algorithm. The algorithm divides a 2D space into 9 parts, using the infinite extensions of the four linear boundaries of the window. Assign a bit-pattern to each region as shown below:

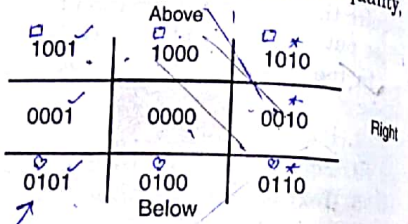


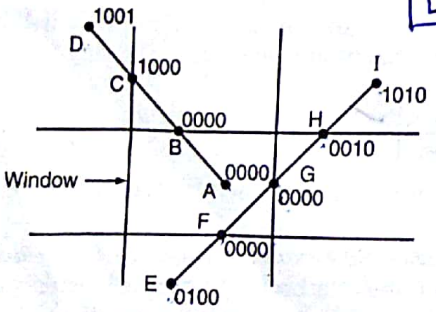
Fig. 9.7.

The numbers in the figure above are called **outcodes**. An outcode is computed for each of the two points in the line. The bits in the outcode represent: **Top, Bottom, Right, Left.** Each bit in the code is set to either a 1 (true) or a 0 (false). If the region is to the *left* of the window, the ***first*** bit of the code is set to 1. If the region is to the *Right* of the window, the *second* bit of the code is set to 1. If to the ***Bottom***, the ***third*** bit is set, and if to the *Top*, the ***fourth*** bit is set. The 4 bits in the code then identify each of the nine regions.

For any endpoint (x, y) of a line, the code can be determined that identifies which region the endpoint lies. The code's bits are set according to the following conditions:

- First bit set 1: Point lies to **left** of window $x < x_{min}$
- Second bit set 1: Point lies to **right** of window $x > x_{max}$
- Third bit set 1: Point lies **below** (bottom) window $y < y_{min}$
- Fourth bits set 1: Point lies **above** (top) window $y > y_{max}$

For example,



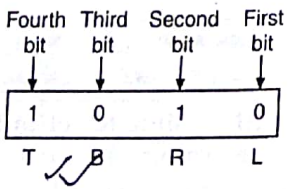


Fig. 9.8.

The outcode 1010 represents a point that is top-right of the viewport. Note that the outcodes for endpoints **must** be recalculated on each iteration after the clipping occurs.

The Algorithm

The Cohen-Sutherland algorithm uses a divide-and-conquer strategy. The line segment's endpoints are tested to see if the line can be trivially accepted or rejected. If the line cannot be trivially accepted or rejected, an intersection of the line with a window edge is determined and the trivial reject/accept test is repeated. This process is continued until the line is accepted.

To perform the trivial acceptance and rejection tests, we extend the edges of the window to divide the plane of the window into the nine regions. Each endpoint of the line segment is then assigned the code of the region in which it lies.

1. Given a line segment with endpoint $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$
2. Compute the 4-bit codes for each endpoint.
   If both codes are **0000**, (bitwise OR of the codes yields 0000) line lies completely **inside** the window: pass the endpoints to the draw routine.
   If both codes have a 1 in the same bit position (bitwise AND of the codes is **not** 0000), the line lies **outside** the window. It can be trivially rejected.
3. If a line cannot be trivially accepted or rejected, at least one of the two endpoints must lie outside the window and the line segment crosses a window edge. This line must be **clipped** at the window edge before being passed to the drawing routine.
4. Examine one of the endpoints, say $P_1 = (x_1, y_1)$. Read P_1's 4-bit code in order: **Left-to-Right, Bottom-to-Top.**
5. When a set bit (1) is found, compute the **intersection I** of the corresponding window edge with the line from P_1 to P_2. Replace P_1 with I and repeat the algorithm.

> **Note:** If both endpoints of a line entirely to one side of the window the line must lie entirely outside of the window. It is trivially rejected and if both end points of a line lie4 inside the window, the entire lie lies inside the window. It is trivially accepted.

Before Clipping

1. Consider the line segment **AD**.
   Point **A** has an outcode of **0000**, and point **D** has an outcode of **1001**. The logical AND of these outcodes is zero; therefore, the line cannot be trivially rejected. Also, the logical OR of the outcodes is no zero; therefore, the line cannot be trivially accepted. The algorithm then chooses **D** as the outside point (its outcode contiain 1's).

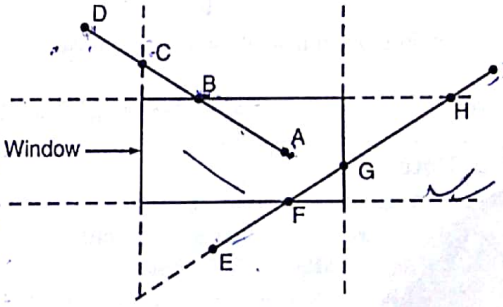


Fig. 9.9.

   By our testing order, we first use the top edge to clip **AD** at **B**. The algorithm then recomputes **B's** outcode as **0000**. With the next iteration of the algorithm, **AB** is tested and is trivially accepted and displayed.
2. Consider the line segment **EI**
   Point **E** has an outcode of **0100**, while point **I's** outcode is **1010**. The results of the trivial tests show that the line can neither be trivially rejected or accepted. Point **E** is determined to be an outside point, so the algorithm clips the line against the

Scanned by CamScanner

bottom edge of the window. Now line EI has been clipped to be line FI. Line FI is tested and cannot be trivially accepted or rejected. Point F has an outcode of 0000, so the algorithm chooses point I as an outside point since its outcodde is 1010. The line FI is clipped against the window's top edge, yielding a new line FH. Line FH cannot be trivially accepted or rejected. Since H's outcode is 0010, the next iteration of the algorthm clips against the window's right edge, yielding line FG. The next iteration of the algorithm tests FG, and it is trivially accepted and display.

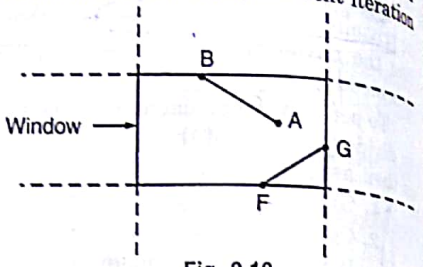


Fig. 9.10.

After Clipping

After clipping the segments AD and EI, the result is that only the line segments AB and FG can be seen in the window.

9.6.2 Line Intersection and Clipping

We determine the intersection points of the lines in category 3 *i.e.* clipping condidate with the boundaries of the window. There intersection points then subdivide the line segment into several line segments which can belong only to visible or nonvisible category *i.e.* category 1 or category 2. The segment in category 1 will be clipped line segment.

The intersection points are found by solving the equations representing the line segment and the boundary lines.

- For left window edge, intersection point will be (x_L, y)

$$y = m(x_L - x_1) + y_1,\ m \neq \infty$$

- For right window edge, intersection point will be (x_R, y)

$$y = m(x_R - x_1) + y_1,\ m \neq \infty$$

- For top window edge, intersection point will be (x, y_T)

$$x = x_1 + \frac{1}{m}(y_T - y_1),\ m \neq 0$$

- For bottom window edge, intersection point will be (x, y_B)

$$x = x_1 + \frac{1}{m}(y_B - y_{1m}),\ m \neq 0$$

Note:
In all above equations
x_L is x-value of left edge of clipping window
x_R is x-value of right edge of clipping window
y_B is y-value of bottom edge of clipping window
y_T is y-value of top edge of clipping window.
and acceptable value of x and y are

$$x_L \le x \le x_R$$
$$y_B \le y \le y_T$$

EXAMPLE 9.2.

Use the cohen Sutherland algorithm to clip line P_1 (70, 20) and P_2 (100, 10) against a window lower left hand corner (50, 10) and upper right hand corner (80, 40).

Solution: $P_1(70, 20)$ and $P_2(100, 10)$
Window lower left corner = (50, 10)
Window upper right corner = (80, 40)

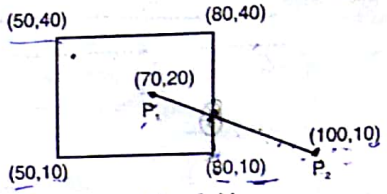


Fig. 9.11.

So, the window is
Now, we assign 4 bit binary outcode.
Point P_1 is inside the window so outcode of P_1 = 0000
and the outcode of P_2 = 0010 as point P_2 is right of window AND of P_1 and P_2

$$\begin{array}{r} 0\ 0\ 0\ 0 \\ .\ 0\ 0\ 1\ 0 \\ \hline 0\ 0\ 0\ 0 \end{array}$$

The result of AND operation is zero so line is partially visible. Slop of line P_1P_2 is m =

$$\frac{y_2 - y_1}{x_2 - x_1} = \frac{10 - 20}{100 - 70} = \frac{-10}{30} = -\frac{1}{3}$$

We have to find intersection of line P_1 P_2 with right edge of window *i.e.* Point P_2.
Let the intersection point be (x, y)
here x = 80, we have to find the value of y.
We use point $P_2(x_2, y_2) = P_2(100, 10)$

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

$$-\frac{1}{3} = \frac{y - 10}{80 - 100}$$

$$y - 10 = \frac{20}{3} \Rightarrow y = \frac{20}{3} + 10 = 16.66.$$

$$\therefore \quad y = 16.66$$

Thus the intersection point P_3 = (80, 16.66)
So, after clipping line P_1P_2 against window, new line is P_1P_3 with co-ordinates P_1(70, 20) and P_3(80, 16.66).

EXAMPLE 9.3.

Given a clipping window A(20, 20) B(60, 20) C(60, 40), D(20, 40). Using Sutherland Cohen algorithm find the visible portion of line segment joining the points P(40, 80) Q(120, 30).

Solution:

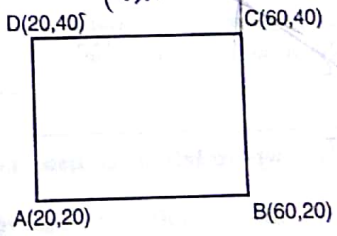


Fig. 9.12.

Scanned by CamScanner

Here $x_L = 20 \quad y_B = 20$

$x_R = 60 \quad y_T = 40$

As we know, the outcodes can be calculated as

Bit 1 = sign of $(y - y_T)$

Bit 2 = sign of $(y_B - y)$

Bit 3 = sign of $(x - x_R)$

Bit 4 = sign of $(x_L - x)$

and sign = 1 if value is +ve and sign = 0 if value is –ve.

Thus, the outcodes of $P(40, 80)$ is 1000

and $Q(120, 30)$ is 0010.

Both endpint codes are not zero and their logical AND is zero. Hence, line cannot be rejected as invisible.

Slope of $PQ(m) = \dfrac{y_2 - y_1}{x_2 - x_1} = \dfrac{30 - 80}{120 - 40} = -\dfrac{5}{8}$

and intersections with window edge are

- **Left** $y = m(x_L - x_1) + y_1 = -\dfrac{5}{8}(20 - 40) + 80$

  $= 92.5$, which is greater than y_T and hence rejected.

- **Right** $y = m(x_R - x_1) + y_1$

  $= -\dfrac{5}{8}(60 - 40) + 80 = 67.5$, which is greater than y_T

  so, it is rejected.

- **Top** $x = x_1 + \dfrac{1}{m}(y_T - y_1) = 40 + \dfrac{1}{-\frac{5}{8}}(40 - 80) = 104,$

  which is greater than x_R and hence rejected.

- **Bottom** $x = x_1 + \dfrac{1}{m}(y_B - y_1) = 40 + \dfrac{1}{-\frac{5}{8}}(20 - 80) = 136,$

  which is greater than x_R and hence rejected.

Since both the values of y are greater than y_T and both are also greater than x_R. Therefore the line segment PQ completely outside the window.

EXAMPLE 9.4.

Let R be the rectangular window whose left-lower hand corner is at $L(-3, 1)$ and upper right-hand corner is at $R(2, 6)$.

(a) Find the region codes for the endpoints in the Fig.

(b) Find the clipping categories for the line segments

(c) Use cohen-sutherland algorithm to clip the line segments.

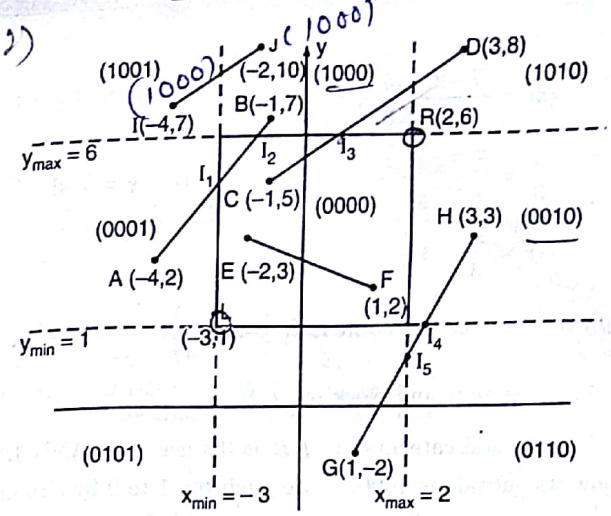


Fig. 9.13.

Solution: (*a*) The region code or outcode for point (x, y) is set according to

$$\left.\begin{aligned} &\text{Bit 1} = \text{sign}\,(y - y_{max}) = \text{sign}\,(y - 6)\\ &\text{Bit 2} = \text{sign}\,(y_{min} - y) = \text{sign}\,(1 - y)\\ &\text{Bit 3} = \text{sign}\,(x - x_{max}) = \text{sign}\,(x - 2)\\ &\text{Bit 4} = \text{sign}\,(x_{min} - x) = \text{sign}\,(-3 - x)\end{aligned}\right\} \text{ and sign } (a) = \begin{cases}1, & \text{if a is +ve}\\ 0 & \text{otherwise}\end{cases}$$

So, $A(-4, 2) \rightarrow 0001 \quad F(1, 2) \rightarrow 0000$

$B(-1, 7) \rightarrow 1000 \quad G(1, -2) \rightarrow 0100$

$C(-1, 5) \rightarrow 0000 \quad H(3, 3) \rightarrow 0010$

$D(3, 8) \rightarrow 1010 \quad I(-4, 7) \rightarrow 1001$

$E(-2, 3) \rightarrow 0000 \quad J(-2, 10) \rightarrow 1000$

(*b*) Now by testing the outcodes, we get the appropriate categories.

Category 1 (Visible): $\overline{EF}$ line segment because the outcodes for both endpoints is 0000

Category 2 (Not Visible): $\overline{IJ}$ line segment, because

1 0 0 1

AND 1 0 0 0

1 0 0 0 (which is not 0000)

Category 3 (Candidates of Clipping):

(1) $\overline{AB}$ line segment since (0001) AND (1000) = 0000

(2) $\overline{CD}$ line segment since (0000) AND (1010) = 0000

(3) $\overline{GH}$ line segment since (0100) AND (0010) = 0000

(c) The clipping lines are $\overline{AB}$, $\overline{CD}$ and $\overline{GH}$.

Scanned by CamScanner

For Line $\overline{AB}$:

$$m = \frac{7-2}{-1+4} = \frac{5}{3}$$

$$\frac{5}{3} = \frac{y-y_2}{x-x_2} \Rightarrow \frac{5}{3} = \frac{y-7}{-3+1} \quad [\because \ x = -3]$$

$$\Rightarrow \qquad y = \frac{11}{3} = 3\frac{2}{3}$$

Thus, the resulting intersection point is $I_1\left(-3, 3\frac{2}{3}\right)$.

Now, we clip $\overline{AI_1}$ segment and work on $\overline{I_1B}$.

The code of I_1 is 0000 and category of $\overline{I_1B}$ is 3 since 0000 AND 1000 = 0000. Now B is outside the window its outcode is 1000, so we push the 1 to 0 by clipping against the line $y_{max} = 6$.

Now $$\frac{5}{3} = \frac{y-y_2}{x-x_2} \Rightarrow \frac{5}{3} = \frac{6-\frac{11}{3}}{x+3} \quad [\because \ y = 6]$$

$$\Rightarrow \qquad 5x + 15 = 7$$

$$\Rightarrow \qquad x = -\frac{8}{5} = -1\frac{3}{5}.$$

So, the resulting intersection point I_2 is $\left(-1\frac{3}{5}, 6\right)$

Thus $\overline{I_2B}$ is clipped. The code for I_2 is 0000. The remaining segment $\overline{I_1I_2}$ is displayed since both endpoints lie in the window.

For Line $\overline{CD}$

The outcode of D is 1010 *i.e.* it is outside the window. We push the first 1 to 0 by clipping against the line $y_{max} = 6$.

Here $$m = \frac{y_2-y_1}{x_2-x_1} = \frac{8-5}{3+1} = \frac{3}{4}$$

and $$y = 6$$

$$\frac{3}{4} = \frac{6-5}{x+1} \Rightarrow x = \frac{1}{3}.$$

So, the resulting intersection point I_3 is $\left(\frac{1}{3}, 6\right)$ and its code is 0000. Thus $\overline{I_3D}$ is clipped and the remaining segment $\overline{CI_3}$ is displayed since it has both endpoints outcodes 0000.

For Line $\overline{GH}$

We can start with either G or H. The code for G is 0100 we push the 1 to a 0 by clipping against the line $y_{min} = 1$. The resulting intersection point is $I_4\left(2\frac{1}{3}, 1\right)$ and its outcode is 0010

we clip $\overline{GI_4}$ and now $\overline{I_4H}$. But segment $\overline{I_4H}$ is not displayed since (0010) AND (0010) = 0010 (which is not 0000)

EXAMPLE 9.5.

Suppose that in an implementation of the Cohen-Sutherland algorithm we choose boundary lines in the top-bottom-right left order to clip a line in category 3, draw a picture to show a worst case scenario, *i.e.*, one that involves the highest number of iterations.

Solution:

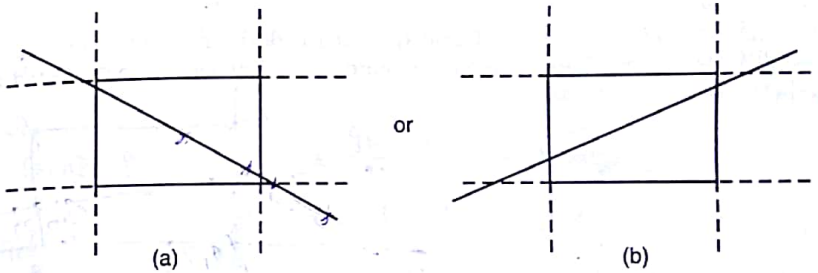


Fig. 9.14.

9.6.3. Mid-point Subdivision Algorithm *(UPTU 2008-09)*

The only difficult part in Cohen-Sutherland algorithm is to find the intersection point of line with window boundary. An alternative to finding itersection points by equation solving is based on bisection method of numerical analysis. The line segment is divided at its midpoint into two smaller line segments. The clipping categories of the two new line segments are then determined. Each segment in category 3 is divided again into smaller segments and categorized. The bisection and categorization process continues until all segments are in category 1 (visible) or category 2 (invisible). The mid-point co-ordinates (x_m, y_m) of a line segment joining $P_1(x_1, y_1)$ to $P_2(x_2, y_2)$ are given by

$$x_m = \frac{x_1 + x_2}{2} \qquad y_m = \frac{y_1 + y_2}{2}$$

We can call mid-point subdivision algorithm as a special case of Cohen-Sutherland algorithm because it uses the same technique as Cohen-Sutherland. It uses the line endpoint outcodes and associated tests to immediately identify which lines are visible and which are invisible. If both the endpoints of a line has outcodes 0000 then the line is totally visible. If both the outcodes are not zero then we have to take logical AND of both outcodes and then we have to decide whether that line is visible or not. If the logical AND result is nonzero, it means the line is totally invisible and if the logical AND is zero, it means a line may be partially visible.

The algorithm is formalized in the following steps:

For each endpoint:

(a) If the endpoint is visible, then it is farthes visible point. The process is complete.

If not, continue.

Scanned by CamScanner

(b) If the line is trivially invisible, no output is generated. The process is complete. If not, continue.

(c) Divide the line P_1P_2 at its midpoint, P_m. Apply the previous tests to the two segments P_1P_m and P_mP_2. If P_mP_2 is rejected as trivally invisible, the midpoint is an overestimation of the farthest visible point. Continue with P_1P_m, otherwise the midpoint is an underestimation of the farthest visible point. Continue with P_2P_m. If the segment becomes so short that the midpoint corresponds to the accuracy of the machine or as specified to the end points, evaluate the visibility of the point and the process is complete.

EXAMPLE 9.6.

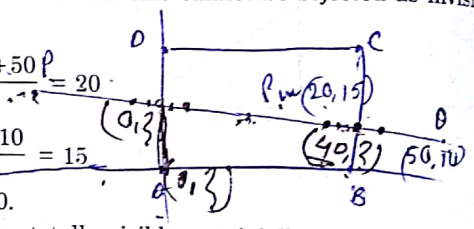
A clipping window ABCD is specified as A(0, 0) B(40, 0), C (40, 40), D(0, 40). Using midpoint subdivision algorithm find the visible portion, if any, of the line segment joining the points P(–10, 20) and Q(50, 10).

Solution: The outcodes of P is 0001 and Q is 0010. Both endpoint codes are not zero and their logical AND is zero, hence we can conclude that line cannot be rejected as invisible.

Now midpoint is

$$x_m = \frac{x_1 + x_2}{2} = \frac{-10+50}{2} = 20$$

$$y_m = \frac{y_1 + y_2}{2} = \frac{20+10}{2} = 15$$



Outcode of midpoint $P_m(x_m, y_m)$ is 0000.

Neither segment PP_m nor P_mQ is either totally visible or trivially-invisible. Lets keep segment PP_m for later processing, and we continue with P_mQ. This subdivision process continues until we find an intersection point with window edge *i.e.*, (40, y). Table shows how the subdivision works.

P	Q	P_m	*Comment*
(– 10, 20)	(50, 10)	(20, 15)	Save PP_m and continue with P_mQ
(20, 15)	(50, 10)	(35, 12)	Continue with P_mQ
(35, 12)	(50, 10)	(42, 11)	Continue with PP_m
(35, 12)	(42, 11)	(38, 11)	Continue with P_mQ
(38, 11)	(42, 11)	(40, 11)	This is the intersection point of line with right window edge.
(– 10, 20)	(20, 15)	(5, 17)	Recall PP_m and continue with PP_m
(– 10, 20)	(5, 17)	(– 3, 18)	Continue with P_mQ
(– 3, 18)	(5, 17)	(1, 17)	Continue with PP_m
(– 3, 18)	(1, 17)	(– 1, 17)	Continue with P_mQ
(– 1, 17)	(1, 17)	(0, 17)	This is the intersection point of line with left window edge.

Thus visible portion of line segment PQ is from (0, 17) to (40, 11).

9.6.4. Liang-Barsky Line Clipping Algorithm

We can write parametric equation of a line segment is in the form

$$x = x_1 + u\Delta x$$

$$y = y_1 + u\Delta y$$

Scanned by CamScanner

or $$\frac{y_T - y_1}{x_R - x_1} < \frac{y_2 - y_1}{x_2 - x_1} < \frac{y_T - y_1}{x_L - x_1}$$

And we clip the entire line if

$$(y_T - y_1)(x_2 - x_1) < (x_L - x_1)(y_2 - y_1)$$

Line Clipping Against Non-Rectangular Clip Windows

In some applications, it is often necessary to clip lines against randomly shaped polygons. Algorithms based on parametric line equations, such as the ***Liang-Barsky method*** can be extended easily convex polygon windows. We do this by modifying the algorithm to include the parametric equations for the boundaries of the clip region. For concave polygon-clipping regions, we can still apply these parametric clipping procedures if we first split the concave polygon into a set of convex polygons.

Clipping algorithms for Circles or other curved-boundary clipping regions are slower because intersection calculations involve non linear curve equations.

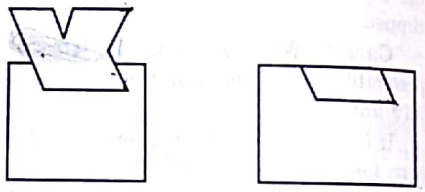
At the first step, lines can be clipped against the bounding rectangle of the curved clipping region. Lines that can be identified as completely outside the bounding rectangle are discarded. To identify inside lines, we can calculate the distance of line endpoints from the circle center. If the square of this distance for both endpoints of a line is less than or equal to the radius squared, we can save the entire line. The remaining lines are then processed through the intersection calculations, which must solve simultaneous circle-line equations.

9.7. Polygon Clipping

Line clipping algorithm connot be used on a polygon because we must generate new edges along the window boundaries as well as clip the original edges. For example.

A polygon is called **convex** if the line joining any two interior points of the polygon lies completely inside the polygon. A non-convex polygon is said to be **concave.**

A polygon with vertices P_1, P_2,.....P_N is said to be **positively oriented** if a tour of the vertices in the given order produces a **counterclockwise circuit.**



(a) Before clipping (b) After clipping

Fig. 9.21.

Let $A(x_1, y_1)$ and $B(x_2, x_2)$ be the endpoints of a directed line segment A point $P(x, y)$ will be to the left of the line segment if the expression

$$C = (x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1)$$ is positive.

We say that the point is to the right of the line segment if this quantity is negative. If a point P is to the right of any one edge of a positively oriented, convex polygon, it is outside the polygon. If it is to the left of every edge of the polygon, it is inside the polygon.

A polygon boundary processed with a line clipper may be displayed as a series of unconnected line segments. For example display of a polygon processed by a line-clipping algorithm is as follows:

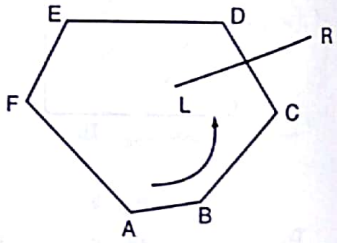


Fig. 9.22.

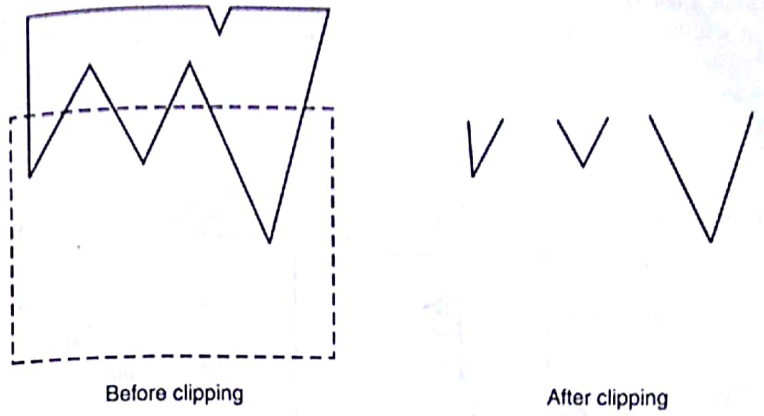


Fig. 9.23.

For polygon clipping, we require an algorithm that will generate one or more closed areas that are then scan converted for the appropriate area fill. Thus, the output of a polygon clipper should be a sequence of vertices that defines the clipped polygon boundaries.

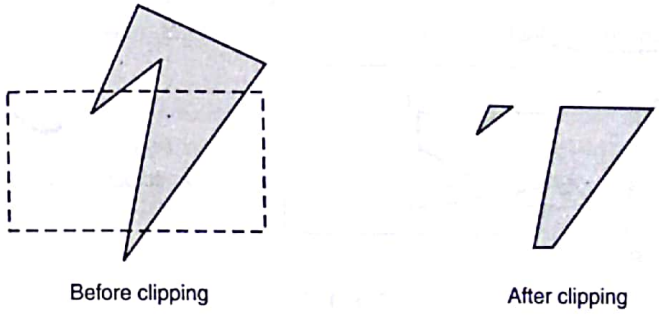


Fig. 9.24. Display of a Correctly Clipped Polygon

In polygon clipping, each edge of the polygon must be tested against each edge of the clip rectangle; new edges must be added and existing edges must be discarded, retained or divided. Multiple polygons may result from clipping a single polygon. We need an organized way to deal with all these cases.

9.7.1. The Sutherland-Hodgman Polygon-Clipping Algorithm *(UPTU 2004)*

Sutherland and Hodgman's polygon-clipping algorithm uses a *divide-and-conquer strategy*: It solves a series of simple and identical problems that, when combined, solve the overall problem.

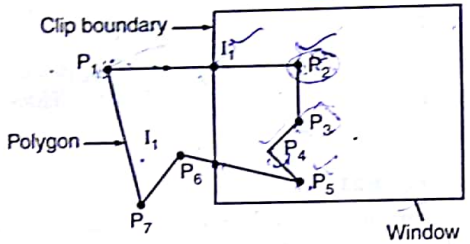
Note the difference between this strategy for a polygon and the Cohen-Sutherland algorithm for clipping a line: The polygon clipper clips against four edges in succession, whereas the line clipper tests the outcodes to see which edge is crossed and clips only when necessary. In Sutherland-Hodgeman algorithm a polygon consists of an ordered sequence of vertices. Let P_1, P_2 ..... P_N be the vertex list of the polygon to be clipped. Let edge E be any edge of the positively oriented convex clipping polygon. The edges will be processed by the clipping algorithm in the order of the vertix pairs.

Scanned by CamScanner

The clipping algorithm will be called once for each vertex of the polygon. For each call the algorithm will return either no vertex at all, the original vertex without change or one or more vertices.

When we are clipping a polygon with respect to any particular edge of the window at that time we have to consider following four different cases:

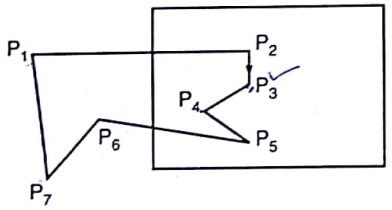
Case-1: If the first vertex is outside the window boundary and the second vertex is inside the window; then the intersection point of polygon with boundary edge of window and the vertex which is inside the window is stored in a output vertex list *i.e.* P_1P_2 edge.



Store I_1 and P_2 in output vertex list.

Fig. 9.25.

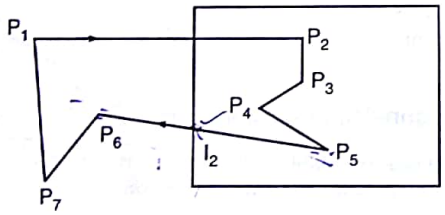
Case-2: If both *i.e.* first and second vertex are inside the window boundary then we have to store the second vertex only in output vertex list, *i.e.* P_2P_3 edge.



Store only P_3 in output vertex list.

Fig. 9.26.

Case-3: If the first vertex is inside the window and second vertex is outside the window boundary then we have to store only intersection point in output vertex list *i.e.* P_5P_6 edge.



Store only I_2 vertex in output vertex list.

Fig. 9.27.

Case-4: If both first and second vertex of a polygon are lieing outside the window boundary then no vertex is stored in output vertex list *i.e.* P_6P_7 edge nothing is stored in output vertex list.

Once all vertices have been considered for one clip window boundary, the output list of vertices is clipped against the next window boundary.

The block diagram for this algorithm is as follows:

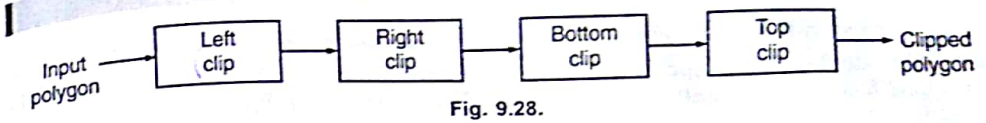


Fig. 9.28.

For example

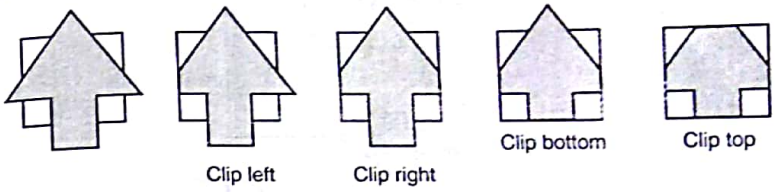


Fig. 9.29.

Flow-chart of Sutherland-Hodgman Algorithm

This algorithm inputs the vertices of a polygon one at a time. For each input vertex, either zero, one or two output vertices are generated depending on the relationship of the input vertices to the clipping edge E.

We denote P as the input vertex, S the previous input vertex and F the first arriving input vertex. The vertex or vertices to be output are determined according to the flow-chart below:

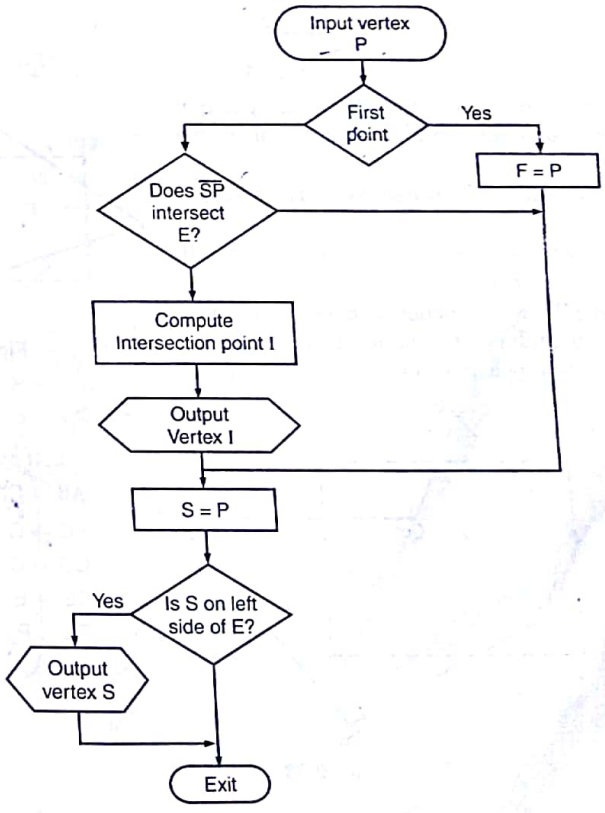


Fig. 9.30.

Scanned by CamScanner

If the polygon has n edges then the edge $\overline{P_n P_1}$ is closing the polygon. In order to avoid the need to duplicate the input of P_1 as the final input vertex the closing logic shown in the flow chart below is called after processing the final input vertex P_n.

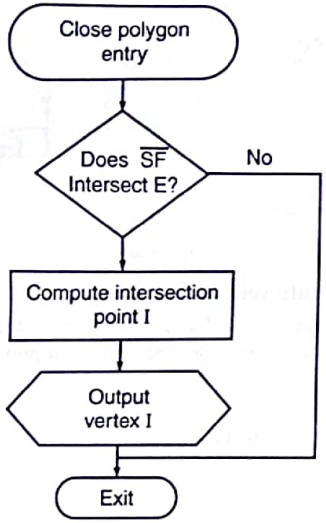


Fig. 9.31.

Let us take an example to understand the polygon clipping by Sutherland Hodgman algorithm. Suppose we are having polygon.

ABCD which we want to clip against a rectangular window.

Here vertex list will be *A, B, C, D, E* and edges will be *AB, BC, CD, DE* and *EA*.

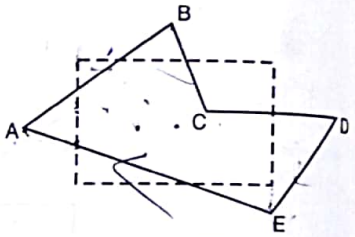


Fig. 9.32.

Step 1. Clip left: We will consider all edges of polygon with respect to left boundary of window. Diagrammatic representation of left clipped polygon is

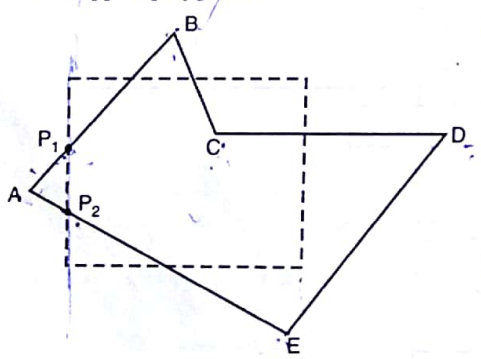
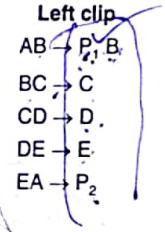


Fig. 9.33.



So after left clipping our output vertex list will become $\{P_1, B, C, D, E, P_2\}$ *i.e.*

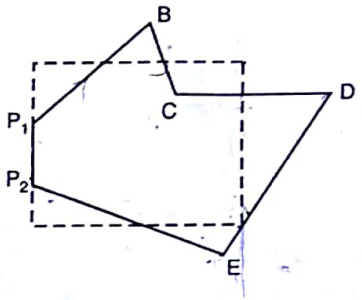
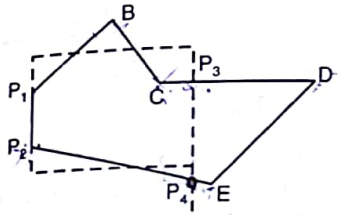


Fig. 9.34.

Step 2. Clip Right: Now modified list of vertices is passed to clip right procedure.



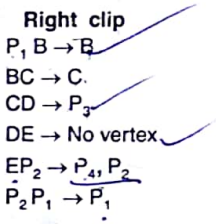


Fig. 9.35.

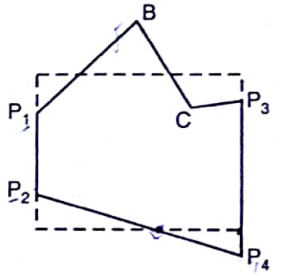
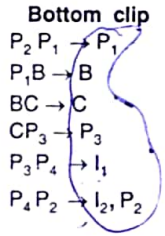


Fig. 9.36.

After right clipping set of vertices will be $\{B, C, P_3, P_4, P_2, P_1\}$

Step-3. Clip Bottom: Now modified list of vertices is passed to this procedure.



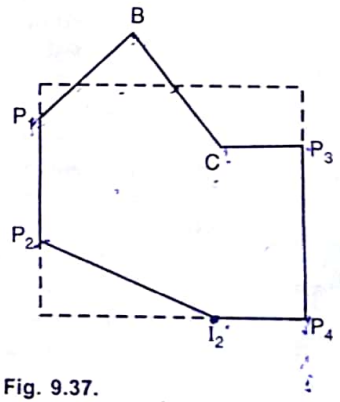


Fig. 9.37.

After bottom clipping set of vertices will be $\{P_1, B, C, P_3, I_1, I_2, P_2\}$

Step 4: Clip top: The modified list of vertices is passed to this procedure.

Scanned by CamScanner

Top clip

$P_1 B \rightarrow I_3$
$BC \rightarrow I_4, C$
$CP_3 \rightarrow P_3$
$P_3 I_1 \rightarrow I_1$
$I_1 I_2 \rightarrow I_2$
$I_2 P_2 \rightarrow P_2$
$P_2 P_1 \rightarrow P_1$

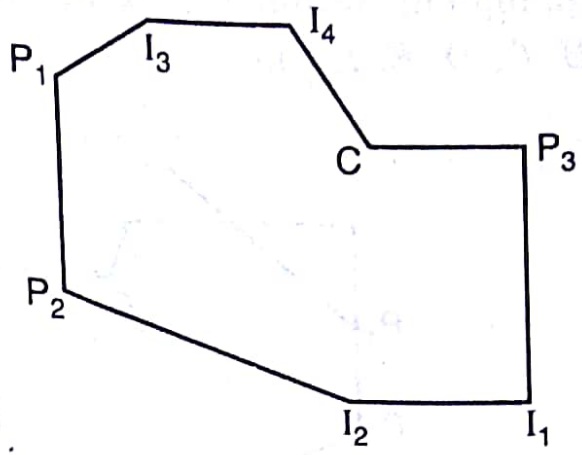


Fig. 9.38.

This is the final clipped polygon.

Limitations with Sutherland-Hodgeman Algorithm

All convex polygons are correctly clipped by the Sutherland-Hodgeman algorithm, but concave polygons may be displayed with extraneous lines. This occurs when the clipped polygon should have two or more separate sections. But since there is only one output vertex list, the last vertex in the list is always joined to the first vertex *i.e.* we are forming an edge between last and first vertex. For example:

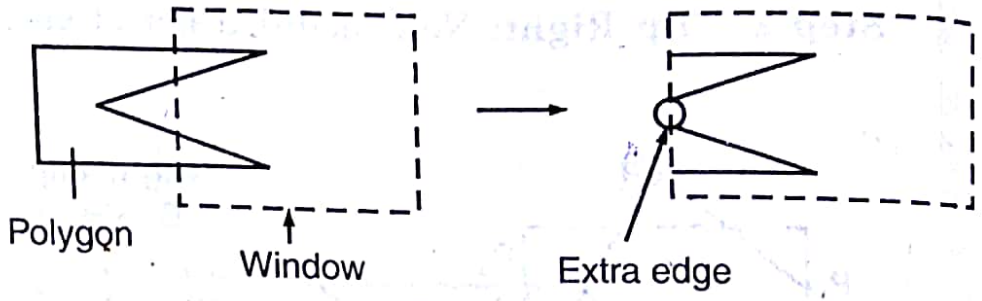


Fig. 9.39.

To overcome this problem one way is to split the concave polygon into two or more convex polygons and process each eonvex polygon separately. Another possibility is to modify the Sutherland-Hodgeman approach to check the final vertex list for multiple vertex points along any clip window boundary and correctly join pairs of vertices. Finally, we could use a more general polygon clipping algorithm such as. Weiler-Atherton polygon clipping algorithm.

Scanned by CamScanner