સ્પર્ધાત્મક પરીક્ષામાં સફળતા મેળવવા
ભારત પબ્લિકેશન્સના સહાયક પુસ્તકો

ભારત પબ્લિકેશન્સ
૧૬, ગોકુલધામ સોસાયટી
જી.આઇ.ડી.સી. રોડ
માંજલપુર, વડોદરા

ISBN 978-93-92795-05-3

Bharat's Computer Graphics Handbook

Computer Graphics Handbook

Dr. Vishwas Raval

Bharat Publications

Computer Graphics Handbook

Dr. Vishwas Raval
Assistant Professor, CSE
Faculty of Technology & Engineering
The M S University of Baroda

Bharat Publications, Vadodara

Computer Graphics Handbook
By - Dr. Vishwas Raval

Publishers
BHARAT PUBLICATIONS
16-B, Gokuldham Society,
G.I.D.C. Road, Manjalpur,
Vadodara

ISBN : 978-93-92795-05-3

First Edition

Price: ₹ 240.00

Printers
NATRAJ OFFSET
KajiMiya's Tekaro, Behind Narayan Kunj
Outside Shahpur Darwaja, Ahmedabad

माँ त्वं ही तारा

ACKNOWLEDGMENTS

First of all, I bow down myself in the lotus feet of the omnipotent, the omnipresent, God and thank God for making me able to do something good for the society. I am grateful to my Guru and father-like **Doctor Uncle** whose blessings have always been with me in every footsteps of my life. Today, whatever I am is their Divine Grace only.

I am thankful to my parents and parents-in-law, for, they have always been my motivation to do good for the society. Especially, my father who has always dreamt of his son doing great and noble! I owe a lot to my lovely wife and daughter as I could not give enough time to them in this duration of writing. Without their generous and selfless support, this work would not have been possible.

Most importantly, this work would not have been possible without the references and resources I used and mentioned in the reference chapter. I am wholeheartedly thankful to all of them who contributed in creating such extremely useful resources which have helped me to carry out this work.

Finally, I thank my colleagues in the department whose support and guidance helped me to move ahead. I am thankful to all my friends, well-wishers, who directly or indirectly supported me throughout this period in all aspects.

TABLE OF CONTENTS

1. INTRODUCTION TO COMPUTER GRAPHICS

Graphics are defined as any sketch or a drawing or a special network that pictorially represents some meaningful information. Computer Graphics is used where a set of images needs to be manipulated or the creation of the image in the form of pixels and is drawn on the computer. Computer Graphics can be used in digital photography, film, entertainment, electronic gadgets, and all other core technologies which are required. It is a vast subject and area in the field of computer science. Computer Graphics can be used in UI design, rendering, geometric objects, animation, and many more. In most areas, computer graphics is an abbreviation of CG. There are several tools used for the implementation of Computer Graphics. The basic is the <graphics.h> header file in Turbo-C, Unity for advanced and even OpenGL can be used for its Implementation.

Computer Graphics refers to several things like the manipulation and the representation of the image or the data in a graphical manner, various technology is required for the creation and manipulation, digital synthesis and its manipulation etc.

There are basically two types of graphics in computers:

Raster Graphics: In raster, graphics pixels are used for an image to be drawn. It is also known as a bitmap image in which a sequence of images is into smaller pixels. Basically, a bitmap indicates a large number of pixels together.

Vector Graphics: In vector graphics, mathematical formulae are used to draw different types of shapes, lines, objects, and so on.

1. APPLICATIONS OF COMPUTER GRAPHICS

Computer graphics deals with creation, manipulation and storage of different type of images and objects.

1. **Computer Art:** Using computer graphics, we can create fine and commercial art which include animation packages, paint packages. These packages provide facilities for designing object shapes and specifying object motion. Cartoon drawing, paintings, logo design can also be done.
2. **Computer Aided Drawing:** Designing of buildings, automobile, aircraft can be done with the help of computer aided drawing. Tthis helps in providing minute details to the drawing and producing more accurate and sharp drawings with better specifications.
3. **Presentation Graphics:** For the preparation of reports or summarizing the financial, statistical, mathematical, scientific, economic data for research reports, managerial reports, creation of bar graphs, pie charts, time chart, can be done using the tools present in computer graphics.
4. **Entertainment:** Computer graphics finds a major part of its utility in the movie industry and game industry. It is used for creating motion pictures, music video, television shows, cartoon animation films. In the game industry where focus and interactivity are the key players. Computer graphics helps in providing such features in the efficient way.

5. **Education:** Computer generated models are extremely useful for teaching huge number of concepts and fundamentals in an easy to understand and learn manner. Using computer graphics, many educational models can be created through which more interest can be generated among the students regarding the subject.
6. **Training:** Specialized system like simulators can be used for training the candidates in a way that can be grasped in a short span of time with better understanding. Creation of training modules using computer graphics is simple and very useful.
7. **Graphic operations:** A general purpose graphics package provides user with varity of function for creating and manipulating pictures. The basic building blocks for pictures are referred to as output primitives. They includes character, string, and geometry entities such as point, straight lines, curved lines, filled areas and shapes defined with arrays of color points. Input functions are used for control & process the various input device such as mouse, tablet etc. Control operations are used for controlling and housekeeping tasks such as clearing display screen etc. All such inbuilt function which we can use for our purpose are known as graphics function.

2. SOFTWARE STANDARD AND GRAPHICS PACKAGES

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

1. **Graphical Kernel System (GKS):** This system was adopted as a first graphics software standard by the international standard organization (ISO) and various national standard organizations including ANSI. GKS was originally designed as the two dimensional graphics package and later extension was developed for three dimensions.
2. **PHIGS (Programmer's Hierarchical Interactive Graphic Standard):** PHIGS is extension of GKS. Increased capability for object modeling, color specifications, surface rendering, and picture manipulation are provided in PHIGS. Extension of PHIGS called "PHIGS+" was developed to provide three dimensional surface shading capabilities not available in PHIGS.

2.1 Graphics Packages:

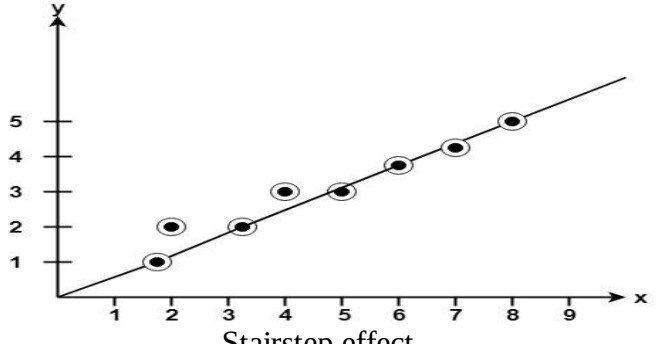
There are mainly two types of graphics packages:

1. **General programming package:** A general programming package provides an extensive set of graphics function that can be used in high level programming language such as C or FORTRAN. It includes basic drawing element shape like line, curves, polygon, color of element transformation etc. Example: − GL (Graphics Library), Open CV etc.
2. **Special-purpose application package:** Special−purpose application package are customized for particular applications which implements required facility and

provides interface so that user need not to worry about how it will work (programming). User can simply use it by interfacing with application. Example: CAD, medical and business systems.

3. OUTPUT PRIMITIVES: POINTS AND LINES

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



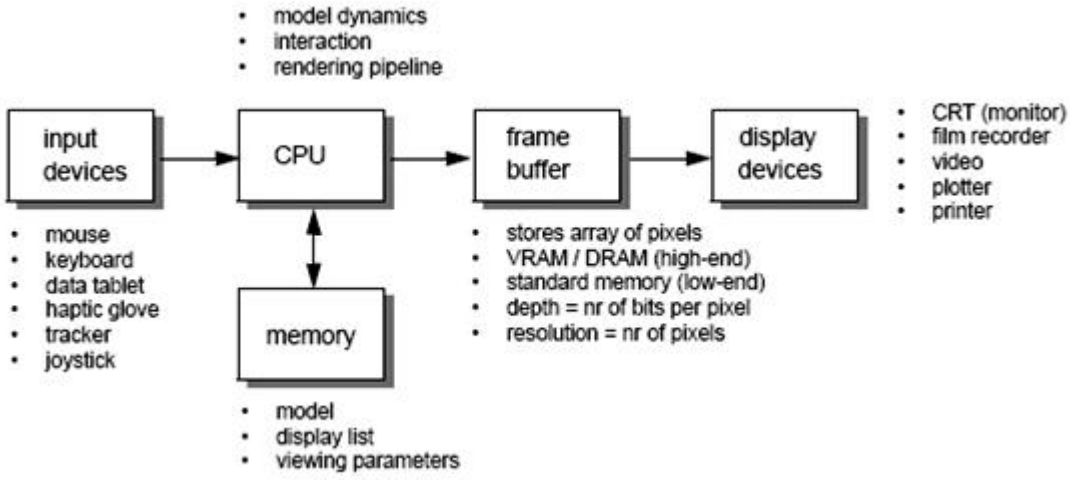
Stairstep effect

12. The stair step shape is noticeable in low resolution system, and we can improve their appearance somewhat by displaying them on high resolution system.

13. More effective techniques for smoothing raster lines are based on adjusting pixel intensities along the line paths.
14. For raster graphics device algorithms discuss in next chapter, object positions are specified directly in integer device coordinates.
15. To load the specified color into the frame buffer at a particular position, we will assume that we have available low−level procedure of the form setpixel(x,y).
16. Similarly for retrieve the current frame buffer intensity we assume to have procedure getpixel(x,y).

4. INPUT DEVICES

The Input Devices are the hardware that is used to transfer transfers input to the computer. The data can be in the form of text, graphics and sound. Output devices display data from the memory of the computer. Output can be text, numeric data, line, polygon, and other objects.



Basic components of Computer Graphics

These Devices include:

1. Keyboard
2. Mouse
3. Trackball
4. Spaceball
5. Joystick
6. Light Pen
7. Digitizer
8. Touch Panels

9. Voice Recognition
10. Image Scanner

1. Keyboard:

Keyboard

The most commonly used input device is a keyboard. The data is entered by pressing the set of keys. All keys are labeled. A keyboard with 101 keys is called a QWERTY keyboard. The keyboard has alphabetic as well as numeric keys. Some special keys are also available.

Numeric Keys: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Alphabetic keys: a to z (lower case), A to Z (upper case)

Special Control keys: Ctrl, Shift, Alt

Special Symbol Keys: ; , " ? @ ~ ? :

Cursor Control Keys: ↑ → ← ↓

Function Keys: F1 F2 F3 F9.

Numeric Keyboard: It is on the right-hand side of the keyboard and used for fast entry of numeric data.

Function of Keyboard: Alphanumeric Keyboards are used in CAD (Computer Aided Drafting)**.** Keyboards are available with special features line screen co-ordinates entry, Menu selection or graphics functions, etc. Special purpose keyboards are available having buttons, dials, and switches. Dials are used to enter scalar values. Dials also enter real numbers. Buttons and switches are used to enter predefined function values.

2. Mouse:

Mouse

A Mouse is a pointing device and used to position the pointer on the screen. It is a small palm size box. There are two or three pressing switches on the top. The movement of the mouse along the x-axis helps in the horizontal movement of the cursor and the movement along the y-axis helps in the vertical movement of the cursor on the screen. The mouse cannot be used to enter text. Therefore, they are used along with a keyboard.

3. Trackball

Trackball

It is a pointing device. It is similar to a mouse. This is mainly used in notebook or laptop computer, instead of a mouse. This is a ball which is half inserted, and by moving fingers on the ball, the pointer can be moved.

4. Space ball:

It is similar to trackball, but it can move in six directions whereas trackball can move in two directions only. The movement is recorded by the strain gauge. Strain gauge is applied with pressure. It can be pushed and pulled in various directions. The ball has a diameter around 7.5 cm. The ball is mounted in the base using rollers. One-third of the ball is an inside box, the rest is outside. It is used for three-dimensional positioning of the object. It is also used to select various functions in the field of virtual reality. It is applicable in CAD applications. Animation is also done using spaceball. It is used in the area of simulation and modeling too.

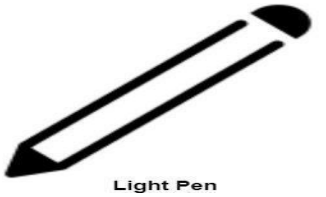
5. Joystick:

A Joystick is also a pointing device which is used to change cursor position on a monitor screen. Joystick is a stick having a spherical ball as it's both lower and upper ends as shown in figure.

The lower spherical ball moves in a socket. The joystick can be changed in all four directions. The function of a joystick is similar to that of the mouse. It is mainly used in Computer Aided Designing (CAD) and playing computer games.

6. Light Pen:

Light Pen (similar to the pen) is a pointing device which is used to select a displayed menu item or draw pictures on the monitor screen. It consists of a photocell and an optical system placed in a small tube. When its tip is moved over the monitor screen, and pen button is pressed, its photocell sensing element detects the screen location and sends the corresponding signals to the CPU.



Light Pen

Light Pens can be used as input coordinate positions by providing necessary arrangements. If background color or intensity, a light pen can be used as a locator. It is used as a standard pick device with many graphics system. It can be used as stroke input devices. It can be used as valuators.

7. Digitizers:

Digitizer

The digitizer is an operator input device, which contains a large, smooth board (the appearance is similar to the mechanical drawing board) & an electronic tracking device, which can be changed over the surface to follow existing lines. The electronic tracking device contains a switch for the user to record the desired x & y coordinate positions. The coordinates can be entered into the computer memory or stored or an off-line storage medium such as magnetic tape.

8. Touch Panels:

Touch Panels is a type of display screen that has a touch-sensitive transparent panel covering the screen. A touch screen registers input when a finger or other object comes in contact with the screen. When the wave signals are interrupted by some contact with the screen that location is recorded. Touch screens have long been used in military applications.

9. Voice Systems (Voice Recognition):

Voice Recognition is one of the newest, most complex input techniques used to interact with the computer. The user inputs data by speaking into a microphone. The simplest form of voice recognition is a one-word command spoken by one person. Each command is isolated with pauses between the words. Voice Recognition is used in some graphics workstations as input devices to accept voice commands. The voice-system input can be used to initiate graphics operations or to enter data. These systems operate by matching an input against a predefined dictionary of words and phrases.

10. Image Scanner

The data or text is written on paper. The paper is fed to scanner. The paper written information is converted into electronic format which is stored in the computer in image format. The input documents can contain text, handwritten material, pictures etc. By storing the document in a computer document became safe for longer period of time. The document will be permanently stored for the future. We can edit and print the document whenever needed. Scanning can be of the black and white or colored picture. On stored picture 2D or 3D rotations, scaling and other operations can also be applied.

5. OUTPUT DEVICES

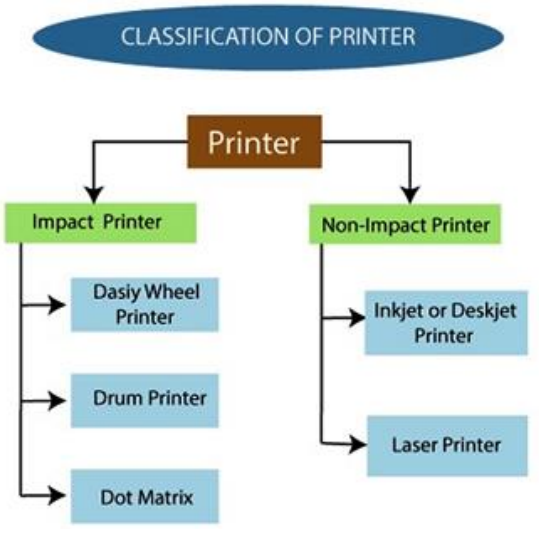
1. Printers:

A printer is a peripheral device which is used to represent the graphics or text on paper. The quality is measured by its resolution. The resolution of any printer is measured in dot per inch (dpi). The printer usually works with the computer and connected via a cable. In present, many digital device support printer features so that we can use Bluetooth, WiFi, and cloud technology to print.

Some types of printers are:

1.1 Impact Printers

1.2 Non-impact Printers



Classification of Printers

1.1 Impact Printers

In impact printers, there is a physical contact established between the print head, ribbon, ink-cartridge and paper. The printers hit print head on an ink-filled ribbon than the letters get printed on the paper. Impact printers work like a typewriter. These printers have three types:

Types of Impact Printers:

1.1.1 Daisy Wheel Printers:

Daisy wheel printer

By these, we can print only one character at a time. The head of this printer looks like a daisy flower, with the printing arms that appear like petals of a flower; that's why it is called "Daisy printer." It can print approx. 90 characters per second. Daisy wheel printers are used to print the professional quality document. It is also called "Letter Quality Printer."

Advantages:

- More reliable
- Better printing Quality

Disadvantages:

- Slow than Dot Matrix
- More Expensive
- Noisy in operation

1.1.2 Drum Printers:

It has a shape like a drum, so it is called **"**Drum Printer." This type of printer contains many characters that are printed on the drum. The surface of the drum is break down into the number of tracks. Total tracks are equal to 132 characters. A drum will have 132 tracks. The number of tracks is divided according to the width of the paper. It can print approx. 150-2500 lines per minute.

Drum Printers

Advantages:

- High Speed
- Low Cost

Disadvantages:

- Poor Printing Quality
- Noisy in Operation

1.1.3 Dot Matrix Printer:

It is also known as the "Impact Matrix Printer." Dot Matrix Printer can print only one character at a time. The dot matrix printer uses print heads consisting of 9 to 24 pins. These pins are used to produce a pattern of dots on the paper to create a separate character. Dot-matrix printer can print any shapes of character, special character, graphs, and charts.

Dot matrix printer

Advantages:

- Low Printing Cost
- Large print size
- Long Life

Disadvantages:

- Slow speed
- Low Resolution

1.2 Non-impact Printers

In Non-impact printers, there is no physical contact between the print head or paper head. A non-impact printer prints a complete page at a time. The Non-impact printers spray ink on the paper through nozzles to form the letters and patterns. The printers that print the letters without the ribbon and on papers are called Non-impact printer. Non-impact printers are also known as "Page Printer."

These printers have two types:

1.2.1 Inkjet Printer:

It is also called "Deskjet Printer**.**" It is a Non-impact printer in which the letters and graphics are printed by spraying a drop of ink on the paper with nozzle head. A Color inkjet printer has four ink nozzles, sapphire, red, yellow, and black, so it is also called CMYK printer. We can produce any color by using these four colors. The prints and graphics of this printer are very clear. These printers are generally used for home purposes.

Deskjet printer

Advantages:

- High-Quality Printout
- Low noise
- High Resolution

Disadvantages:

- Less Durability of the print head
- Not suitable for high volume printing
- Cartridges replacement is expensive

1.2.2 Laser Printer:

It is also called **"**Page Printer**"** because a laser printer process and store the whole page before printing it. The laser printer is used to produce high-quality images and text. It is used with personal computers generally. The laser printers are mostly preferred to print a large amount of content on paper.

Laser printer

Advantages:

- High Resolution
- High printing Speed
- Low printing Cost

Disadvantages:

- Costly than an inkjet printer
- Larger and heavier than an inkjet printer

2. Plotters:

A plotter is a special type of output device. It is used to print large graphs, large designs on large papers. For Example, Construction maps, engineering drawings, architectural plans, and business charts etc. It is similar to a printer, but it is used to print vector graphics. Following are the types of Plotter:

2.1 Flatbed Plotter:

In a flatbed plotter, the paper is kept in a stationary position on a table or a tray. A flatbed plotter has more than one pen and a holder. The pen rotates on the paper upside-down and right-left by a motor. Every pen has a different color ink, which is used to draw the multicolor design. We can quickly draw Cars, Ships, Airplanes, Dress design, road and highway blueprints, etc. using such printers.

Flatbed plotter

Advantages:

- Larger size paper can be used
- Drawing Quality is similar to an expert

Disadvantages:

- Slower than printers
- More Expensive than printers
- Do not produce high-Quality text printouts

2.2 Drum Plotter:

It is also called "Roller plotter." There is a drum in this plotter. We can apply the paper on the drum. When the plotter starts, these drums moves back and forth, and the image is drawn. Drum plotter has more than one pen and penholders. The pens easily moves right to left and left to right. The movement of pens and drums are controlled by graph plotting program. It is used in industry to produce large drawings (up to A0).

Drum plotter

Advantages:

- Draw Larger Size image
- We can print unlimited length of the image

Disadvantages:

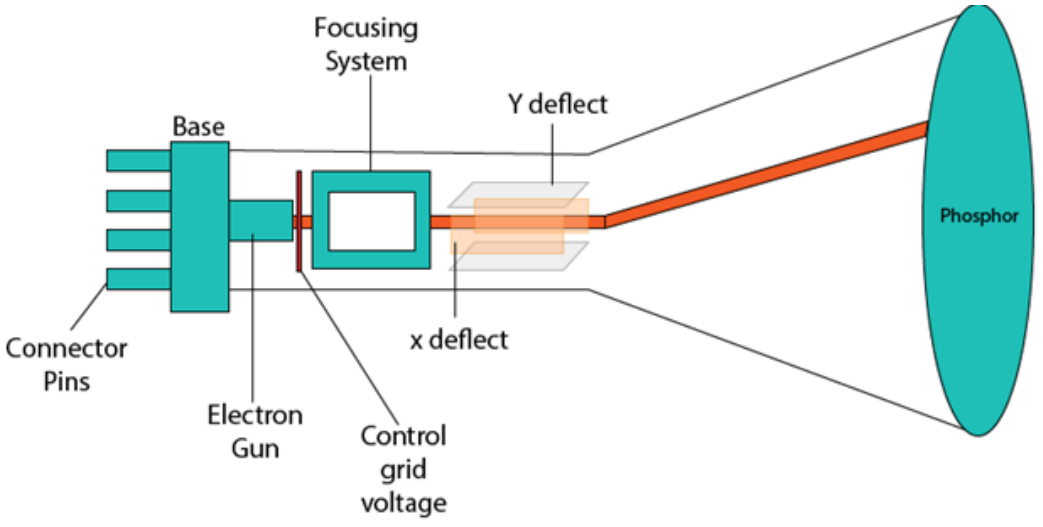
- Very costly

6. VISUAL DISPLAY DEVICES

The primary output device in a graphics system is a video monitor. Although many technologies exist, but the operations of most video monitors are based on the standard Cathode Ray Tube (CRT) design.

1. Cathode Ray Tubes (CRT)

A cathode ray tube (CRT) is a specialized vacuum tube in which images are produced when an electron beam strikes a phosphorescent surface. It modulates, accelerates, and deflects electron beam(s) onto the screen to create the images. Most desktop computer displays make use of CRT for image displaying purposes.



Typical CRT Structure

Construction of a CRT:

1. The primary components are the heated metal cathode and a control grid.
2. The heat is supplied to the cathode by passing current through the filament.
3. This way the electrons get heated up and start getting ejected out of the cathode filament.
4. This stream of negatively charged electrons is accelerated towards the phosphor screen by supplying a high positive voltage.
5. This acceleration is generally produced by means of an accelerating anode.
6. The Focusing System is used to force the electron beam to converge to small spot on the screen.
7. If there will not be any focusing system, the electrons will be scattered because of their own repulsions and hence we won't get a sharp image of the object.
8. This focusing can be either by means of electrostatic fields or magnetic fields.

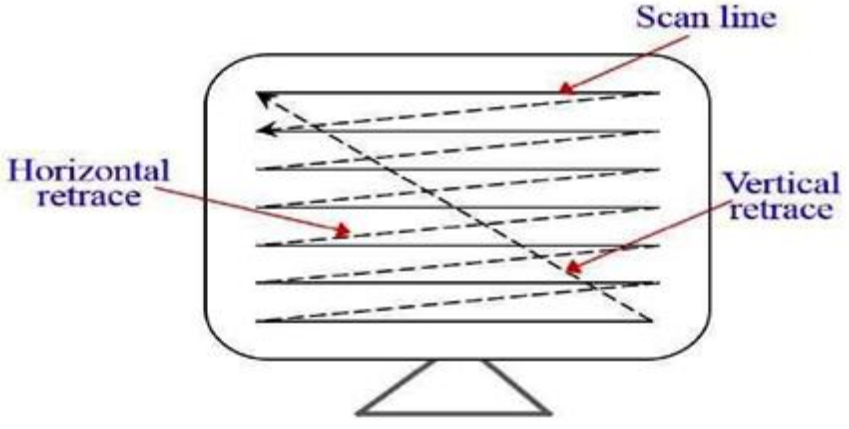
The electron beam (cathode rays) passes through a highly positively charged metal cylinder that forms an electrostatic lens. This electrostatic lens focuses the cathode rays to the center of the screen in the same way like an optical lens focuses the beam of light. Two pairs of parallel plates are mounted inside the CRT. From these two pairs, one pair is mounted on the top and bottom of the CRT tube, and the other pair on the two opposite sides. The magnetic field produced by both these pairs is such that a force is generated on the electron beam in a direction which is perpendicular to both the direction of magnetic field, and to the direction of flow of the beam. One pair is mounted horizontally and the other vertically.

Different kinds of phosphors are used in a CRT. The difference is based upon the time for how long the phosphor continues to emit light after the electron beam disappears. This

property of staying electron on screen is referred to as Persistence. The number of points displayed on a CRT is referred to as resolutions (eg. 1024x768).

2. Raster-Scan

In this system, a beam of an electron is moved across the screen. It moves from top to bottom considering one row at a time. As the beam of electron moves through each row, its intensity is alternatively turned on and off which helps to create a pattern of spots that are illuminated. When each scan of the line is refreshed it returns to the left side of the screen. This motion is known as Horizontal retrace. As a particular frame ends, the beam of electron moves to the left top corner of the screen to move to another frame. This motion is referred to as Vertical retrace. Each complete scanning of a screen is normally called a frame. The frame rate, normally 24 fps, is rate at which frames are sent for display. The Refresh rate of your display refers to how many times per second the display is able to draw a new image. This is measured in Hertz (Hz). For example, if your display has a refresh rate of 144Hz, it is refreshing the image 144 times per second.



Raster Scan Displays

Picture definition is stored in a memory area called the frame buffer. This frame buffer stores the intensity values for all the screen points. Each screen point is called a pixel (picture element or pixel). In black and white systems, the frame buffer storing the values of the pixels is called a bitmap. Each entry in the bitmap is a 1-bit data which determines the on (1) and off (0) of the intensity of the pixel.

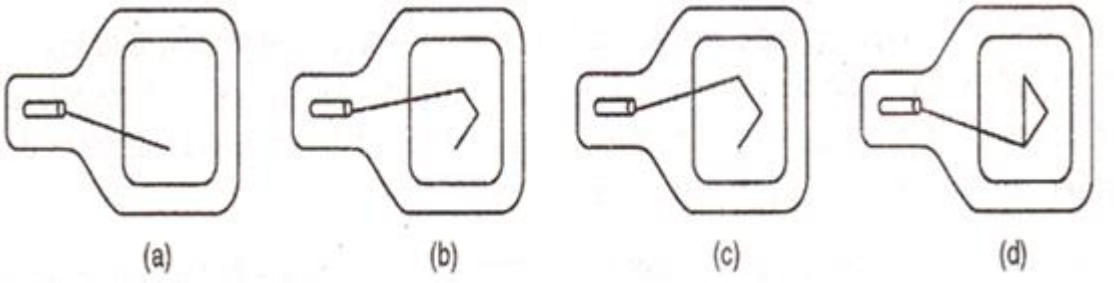
In color systems, the frame buffer storing the values of the pixels is called a pixmap (Though nowadays many graphics libraries refere it as bitmap only). Each entry in the pixmap occupies a number of bits to represent the color of the pixel. For a true color display, the number of bits for each entry is 24 (8 bits per red/green/blue channel, each channel 28 =256 levels of intensity value, ie. 256 voltage settings for each of the red/green/blue electron guns).

Aspect ratio is number of pixels in vertical direction to number of pixels in horizontal direction. A flicker is the appearance of flashing or unsteadiness in an image on a display

screen. This issue can occur when the video refresh rate is too low, other video related issues, and in some cases hardware issues with the monitor. Commonly, when this issue occurs, users will experience more eyestrain.

3. Random-Scan (Vector Display) or Stroke-Writing or Calligraphic displays:

The CRT's electron beam is directed only to the parts of the screen where a picture is to be drawn. The picture definition is stored as a set of line-drawing commands in a refresh display file or a refresh buffer in memory.



Random Scanning Displays

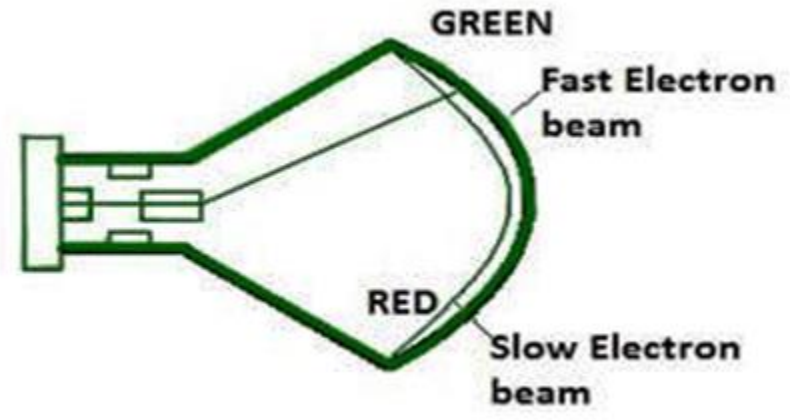
Random-scan generally have higher resolution than raster systems and can produce smooth line drawings, however it cannot display realistic shaded scenes.

4. Color CRT Monitors:

The CRT Monitor display by using a combination of phosphorus with different colors. There are two popular approaches for producing color displays with a CRT are:

- Beam Penetration Method
- Shadow-Mask Method

4.1 Beam Penetration Method:



Beam Penetration Model Structure

The Beam-Penetration method are used with random-scan monitors. In this method, the CRT screen is coated with two layers of phosphorus, red and green and the displayed color depends on how far the electron beam penetrates the phosphorus layers. This method produces four colors only, red, green, orange and yellow. A beam of slow electrons excites the outer red layer only; hence screen shows red color only. A beam of high-speed electrons excites the inner green layer and the screen shows a green color.

Advantages:

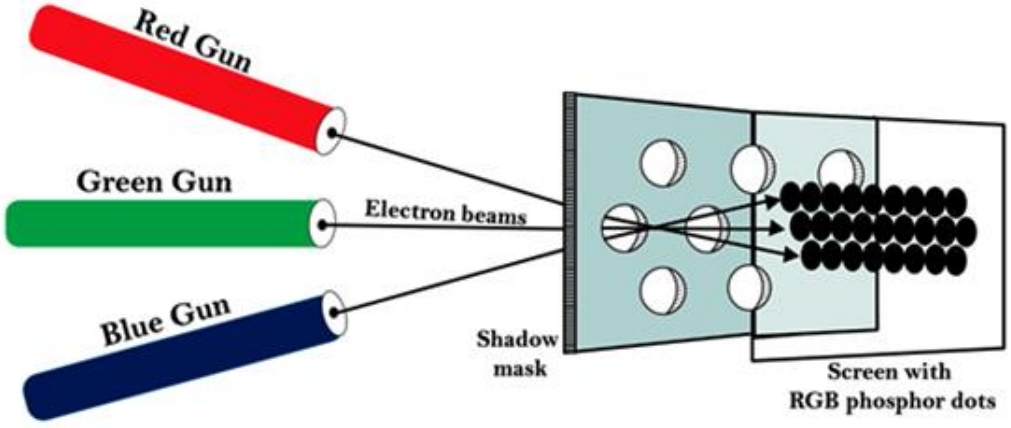
- Inexpensive

Disadvantages:

- Only four colors are possible
- Quality of pictures is not as good as with another method.

4.2 Shadow-Mask Method:

Shadow Mask Method is commonly used in Raster-Scan System because they produce a much wider range of colors than the beam-penetration method. It is used in the majority of color TV sets and monitors.



Shadow Mask Method

Construction:

A shadow mask CRT has 3 phosphorus colored with red, blue and green dots at each pixel position emitting red, green and blue light. This type of CRT has 3 electron guns, one for each color dot and a shadow mask grid just behind the phosphor coated screen. Shadow mask grid is pierced with small round holes in a triangular pattern.

The deflection system of the CRT operates on all 3 electron beams simultaneously. The 3 electron beams are deflected and focused as a group onto the shadow mask, which contains a sequence of holes aligned with the phosphor- dot patterns. When the three beams pass through a hole in the shadow mask, they activate a dotted triangle, which occurs as a small color spot on the screen. The phosphor dots in the triangles are organized so that each

electron beam can activate only its corresponding color dot when it passes through the shadow mask.

Advantages:

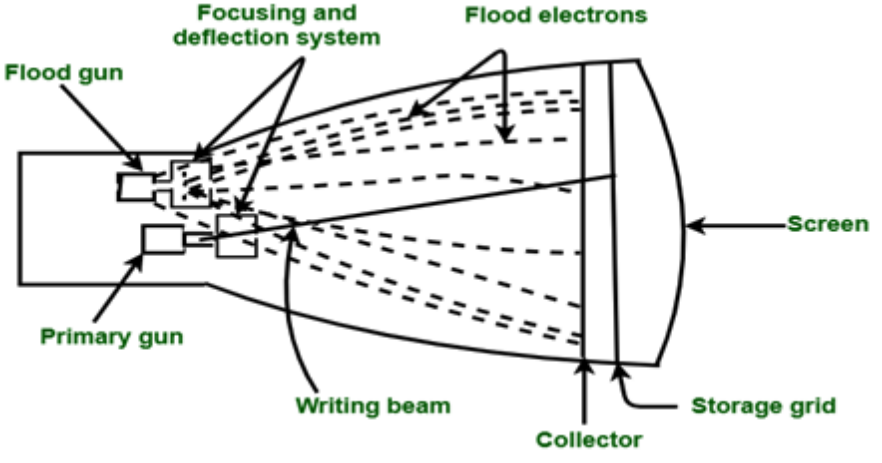
- Realistic image
- Million different colors to be generated
- Shadow scenes are possible

Disadvantages:

- Relatively expensive compared with the monochrome CRT.
- Relatively poor resolution
- Convergence Problem

5. Direct View Storage Tubes:

DVST terminals also use the random scan approach to generate the image on the CRT screen. The term "storage tube" refers to the ability of the screen to retain the image which has been projected against it, thus avoiding the need to rewrite the image constantly.



DVST Structure

There are two guns used in DVST: Primary guns are used to store the picture patter and the Flood gun or Secondary gun are used to maintain picture display.

Advantages:

- No refreshing is needed.
- High Resolution
- Cost is very less

Disadvantages:

- It is not possible to erase the selected part of a picture.

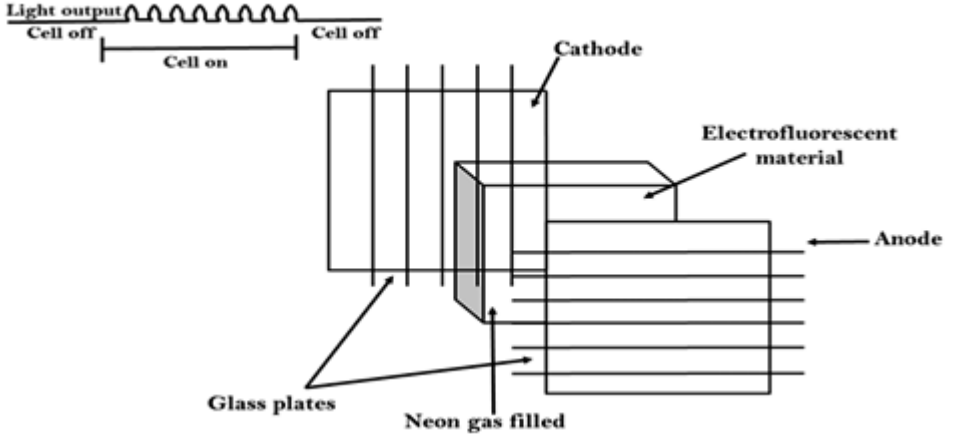
- It is not suitable for dynamic graphics applications.
- If a part of picture is to modify, then time is consumed.

6. Flat Panel Display:

The Flat-Panel display refers to a class of video devices that have reduced volume, weight and power requirement compared to CRT. For example, Small T.V. monitor, calculator, pocket video games, laptop computers, an advertisement board in elevator. There are three types of displays: Plasma, LEDs and LCDs.

6.1 Plasma Panel Display:

Plasma-Panels are also called as Gas-Discharge Display. It consists of an array of small lights. Lights are fluorescent in nature.



A Plasma Panel Display

The essential components of the plasma-panel display are as follows:

Cathode consists of wires and delivers negative voltage to gas cells. The voltage is released along with the negative axis. Anode also consists of wires and delivers positive voltage. The voltage is supplied along positive axis. Fluorescent cells consist of small pockets of gas liquids. When the voltage is applied to this liquid (neon gas), it emits light. The Glass Plates plates act as capacitors. Till the voltages are applied, the cell glows continuously. The gas will be slower when there is a significant voltage difference between horizontal and vertical wires. The voltage level is kept between 90 volts to 120 volts. Plasma display does not require refreshing. Erasing is done by reducing the voltage to 90 volts.

Each cell of plasma has two states and hence the cell is said to be stable. Displayable point in plasma panel is made by the crossing of the horizontal and vertical grid. The resolution of the plasma panel can be up to 512 * 512 pixels.

Advantages:

- High Resolution
- Large screen size is also possible.

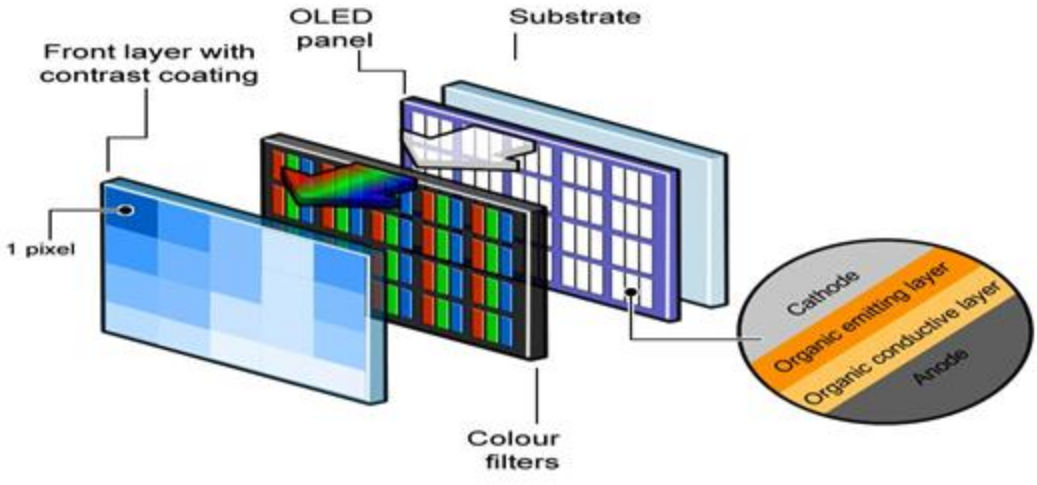
- Less volume
- Less weight
- Flicker Free Display

Disadvantages:

- Poor Resolution
- Wiring requirement anode and the cathode is complex.
- Its addressing is also complex.

6.2 Emissive Display (LED-Light Emitting Diode)

The emissive displays are devices that convert electrical energy into light. Examples are Plasma Panel discussed above, thin film electroluminescent display and LED (Light Emitting Diodes). In an LED, a matrix of diodes is organized to form the pixel positions in the display and picture definition is stored in a refresh buffer. Data is read from the refresh buffer and converted to voltage levels that are applied to the diodes to produce the light pattern in the display.



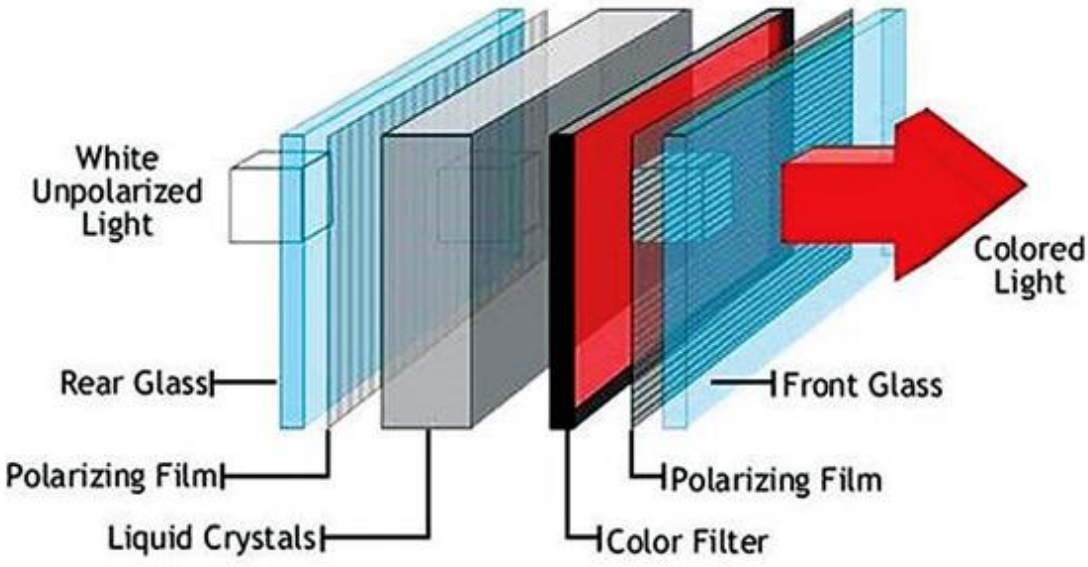
LED Displays

6.3 Non-Emissive Display (LCD-Liquid Crystal Display)

The Non-Emissive displays use optical effects to convert sunlight or light from some other source into graphics patterns. Examples are LCD (Liquid Crystal Device). Liquid Crystal Displays are the devices that produce a picture by passing polarized light from the surroundings or from an internal light source through a liquid-crystal material that transmits the light.

LCD uses the liquid-crystal material between two glass plates; each plate is the right angle to each other and liquid is filled between plates. One glass plate consisting of rows of conductors arranged in vertical direction and another glass plate is consisting of a row of

conductors arranged in horizontal direction. The pixel position is determined by the intersection of the vertical & horizontal conductors. This position is an active part of the screen. Liquid crystal display is temperature dependent. It is between zero to seventy degree Celsius. It is flat and requires very little power to operate.



Modern LCD Displays

Advantages:

- Low power consumption.
- Small Size
- Low Cost

Disadvantages:

- LCDs are temperature-dependent (0-70°C)
- LCDs do not emit light; as a result, the image has very little contrast.
- LCDs have no color capability.
- The resolution is not as good as that of a CRT.

7. FRAME BUFFER

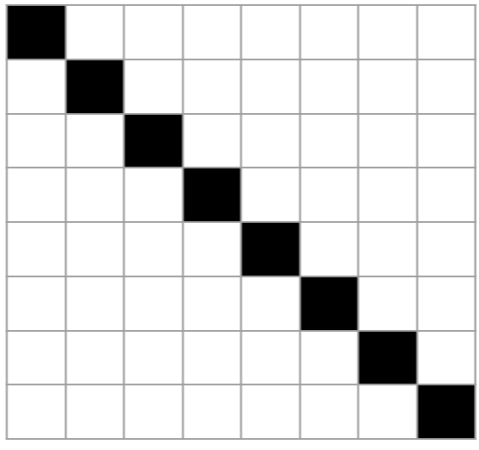
In the previous sections, we talked about images displayed via memory to various types of displays. So how is the image stored inside computer memory? The answer is Frame Buffer. Frame buffer is an important component of computer graphics. There is a display controller that retrieves the contents from the frame buffer and displays the image accordingly. It is a large contiguous piece of memory where the image is stored as a pattern

of bits. Actually, a frame buffer is a portion of Random access Memory (RAM). However, Modern video cards contain frame buffer circuitry in their cores.

As we know that computers understand only the binary language and everything is stored in the form of bits even if it is an image. At a minimum, there is one memory bit for each pixel. This is called the bit plane. That is the number of bits required to store the image equal to the number of pixels. Amount of frame buffer memory required depends on the resolution of the screen. One memory bit can have 2 possible states. A single bit plane will give a black and white display. That is, zero is for black and one is for white. Now, suppose you want to draw a straight line then put the values for those pixels as one and the remaining will be stored as zero.

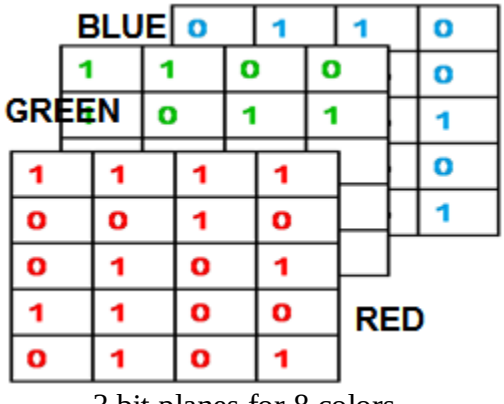
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

Image in the form of Pattern of bits



Pixels in 8X8 Screen

Suppose, we want to change the image then we only need to modify the contents of the frame buffer to represent new patterns of pixels with multiple colors. Colors are incorporated using additional bit planes. That is for colors, we need to store more information per pixel. Therefore, there will be more bits corresponding to each pixel location on the screen. These are called additional bit planes. Suppose we have 3 bits per pixel. Hence, we can have 23 combinations with 8 possible values in each bitplane. That is, we can interpret 8 intensity levels.



3 bit planes for 8 colors

Color	R	G	B
Black	0	0	0
Red	1	0	0
Green	0	1	0
Blue	0	0	1
Yellow	1	1	0
Cyan	0	1	1
Magenta	1	0	1
White	1	1	1

With n bit planes a total of 2^n intensity levels are possible. The memory required by the frame buffer depends on the Total number of bit planes/number of colors/total intensity levels and the Resolution of the screen.

Memory required by frame buffer=n x resolution

Example: There is a system with resolution 640 X 480. Tell the size of the frame buffer to store 12 bits per pixel?

Resolution = 640 X 480

Number of bits/pixel = N = 12

Required Frame Buffer Memory = N X Resolution

= 12 X 640 X 480

Memory in bytes = 12 x 640 X 480 / 8

= 460800 Bytes

Memory in KB = 460800/1024

= 450 KB

EXERCISE

1. What is computer graphics? Why do we study it? What are the different applications of computer graphics in various domains?
2. What are the display devices? Explain them in details. Which output device can give clearer picture?
3. Write a short note on different output devices.
4. List various input devices and explain them in details.
5. Calculate the size of frame buffer to store 16 bits per pixel if the screen resolution is 1280X960.

2. RASTER ALGORITHMS

1. SCAN CONVERSION

It is a process of representing graphics objects as a collection of pixels. The graphics objects are continuous whereas the pixels used are discrete. Each pixel can have either on or off state. The process of converting pixel into objects is also called as rasterization.

The circuitry of the video display device of the computer is capable of converting binary values (0, 1) into a pixel on or off. Pixel off is represented by value 0 and on is represented using value 1. Using this ability computer represents graphics picture with discrete dots. Any model of graphics can be reproduced with a dense matrix of dots or points. Graphics objects are perceived as collection of points, lines, circles, ellipses and polygons. For generating graphical object, many algorithms have been developed which will be discussed in this chapter. The advantages of developing algorithms for scan conversion are they can generate graphics objects at a faster rate, memory can be used efficiently and a higher level of graphical objects can be developed. Examples of objects which can be scan converted are point, line, sector, arc, ellipse, rectangle, polygon, characters, filled Regions etc.

2. LINE

A line connects two points. It is a basic element in graphics. To draw a line, you need two points between which you can draw a line. In the following three algorithms, we refer the one point of line as X0,Y0 and the second point of line as X1,Y1.

2.1 DDA Algorithm

Digital Differential Analyzer DDA algorithm is the simplest line generation algorithm given below:

1. Get the input of two end points (X0,Y0) and (X1,Y1).
2. Calculate the difference between two end points.

   $\Delta x = X1 - X0$

   $\Delta y = Y1 - Y0$
3. Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If $|\Delta x| > |\Delta y|$, then you need more steps in x coordinate; otherwise in y coordinate.

   ```
   if (|Δx| > |Δy|)
      Steps = |Δx|;
   else
      Steps = |Δy|;
   ```
4. Calculate the increment in x coordinate and y coordinate.

   $X_{increment} = \Delta x / Steps;$

$Y_{increment} = \Delta y / Steps;$

5. Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

```
for(int v=0; v < Steps; v++)
{
  Xk+1 = Xk + Xincrement
  Yk+1 = Yk + Yincrement
  putpixel(Round(x), Round(y));
}
```

Advantages:

- It is a simple algorithm to implement.
- It is a faster algorithm than the direct line equation.
- We cannot use the multiplication method in Digital Differential Analyzer.
- Digital Differential Analyzer algorithm tells us about the overflow of the point when the point changes its location.

Disadvantages:

- The floating-point arithmetic implementation of the Digital Differential Analyzer is time-consuming.
- The method of round-off is also time-consuming.
- Sometimes the point position is not accurate.

Example: Calculate the points between the points (5, 6) and (13, 10).

Solution:

Step-01:

(X0, Y0) = (5, 6)

(Xn, Yn) = (13, 10)

Step-02: Calculate ΔX, ΔY and M from the given input.

ΔX = Xn – X0 = 13 – 5 = 8

ΔY =Yn – Y0 = 10 – 6 = 4

M = ΔY / ΔX = 4 / 8 = 0.50

Step-03: Calculate the steps.

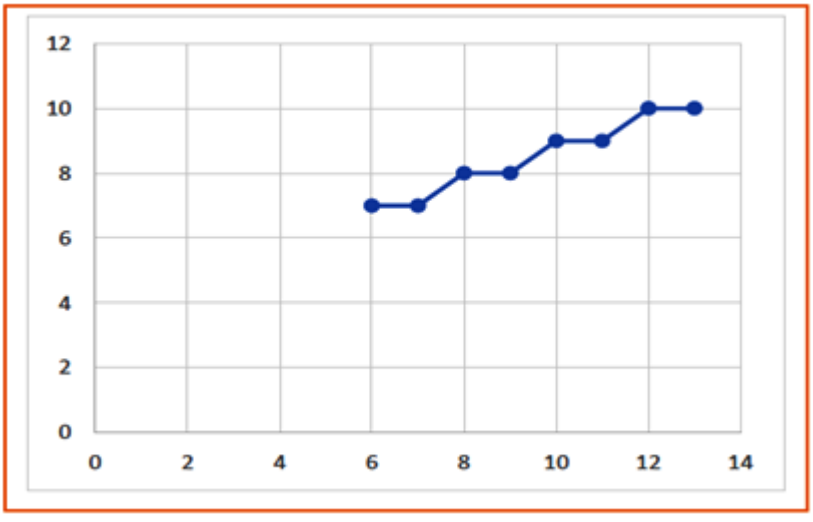
As |ΔX| > |ΔY| = 8 > 4, so Steps = |ΔX| = 8

Step-04:

X_{incr}= ΔX/Steps

Y_{incr}= ΔY/Steps

X_{k+1}	Y_{k+1}	Round off (X_{k+1}, Y_{k+1})
5	6	(5, 6)
6	6.5	(6, 7)
7	7	(7, 7)
8	7.5	(8, 8)
9	8	(9, 8)
10	8.5	(10, 9)
11	9	(11, 9)
12	9.5	(12, 10)
13	10	(13, 10)



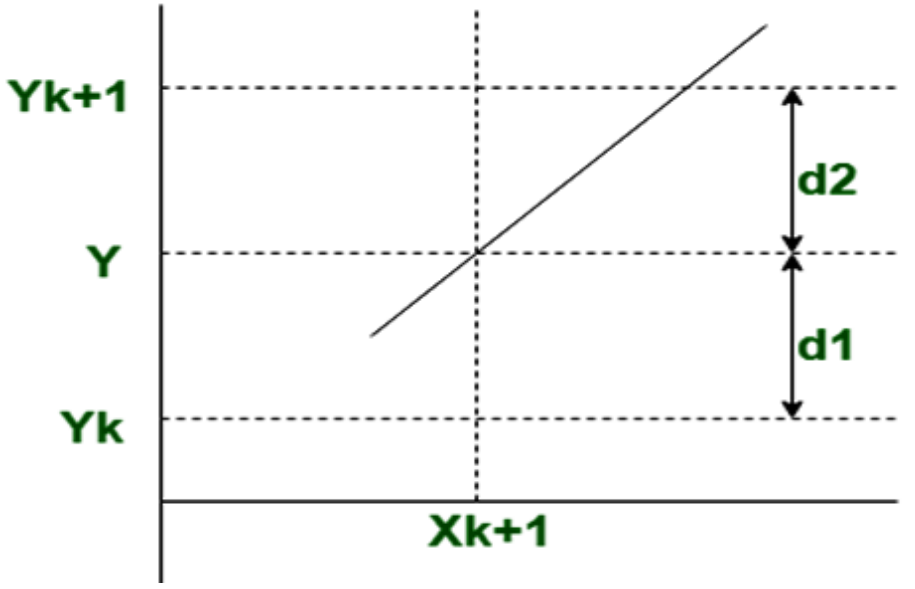
2.2 Bresenham Line Drawing Algorithm

Suppose we have to draw a line with coordinates (x1, y1) and (x2, y2). To draw the line using Bresenham's line drawing algorithm, first of all, calculate the slope of the line from the given coordinates by using,

$$m = \Delta y/\Delta x \text{ where, } \Delta x = x2 - x1 \; \Delta y = y2 - y1$$

There can be three values of slope (m) i.e. m < 1 or m > 1 or m = 1. On the basis of the value of the slope, the decision parameter is calculated which gives the decision about the

selection of the next pixel point which has the least distance from the true line. We'll derive the decision parameter P_k for slope ($m < 1$), to understand it better.



2.2.1 Derivation of the decision parameter P_k ($m < 1$)

Let us consider a straight line passing through a point $A(x_k + 1, y)$,

Where, $x_{k+1} = x_k + 1$ and y satisfies the equation of the line i.e.

$$y = m (x_k + 1) + b \qquad \text{------(1)}$$

The next pixel on the line will be either to $y_k + 1$ or y_k, vertically, because $\Delta y < \Delta x$.

If we choose the top pixel, the points in d2 region,

$$x_{k+1} = x_k + 1 \quad \text{and} \quad y_{k+1} = y_k + 1$$

If we choose bottom pixel, the points in d1 region,

$$x_k + 1 = x_k + 1 \quad \text{and} \quad y_{k+1} = y_k$$

So from the figure given,

$$d2 = (y_k + 1) - y = y_k + 1 - m (x_k + 1) - b$$

$$d1 = y - y_k = m (x_k + 1) + b - y_k$$

Now, we calculate the values of $d_1 - d_2$

$d1 - d2 = m (x_k + 1) + b - y_k - y_k - 1 + m (x_k + 1) + b$ ----(2)

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

$d1 - d2 = 2\Delta y (x_k + 1) / \Delta x + 2b - 2y_k - 1$

To remove Δx from denominator, multiply Δx on both sides;

$\Delta x (d1 - d2) = \Delta x (2\Delta y (x_k + 1) / \Delta x + 2b - 2y_k - 1)$

Let $P_k = \Delta x\ (d1 - d2)$, thus introducing the decision variable

$P_k = 2\Delta y\ x_k - 2\Delta x\ y_k + c$ where $c = 2\Delta y + \Delta x\ (2b - 1)$

Let us calculate next decision variable:

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

Now, solve the difference $P_{k+1} - P_k$

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

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

Since $x_{k+1} = x_k + 1$ always,

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

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

Now, If $P_k < 0$ i.e. d1<d2 (because Δx is considered as absolute) then,

$y_{k+1} = y_k$ (We will choose the nearest y_k pixel)

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

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

Now, If $P_k > 0$ i.e. d1>d2 Then,

$y_{k+1} = y_k + 1$(We will choose the nearest $y_k + 1$ pixel)

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

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

Now, we will use P_k to get initial parameter P_0 using (x1,y1)

$P_0 = 2\Delta y\ x_1 - 2\Delta x\ y_1 + 2\Delta y + \Delta x\ (2b - 1)$

put $b = y1 - m\ x1$ where $m = \Delta y/\Delta x$

Hence, $P_0 = 2\Delta y\ x_1 - 2\Delta x\ y_1 + 2\Delta y + \Delta x\ (2(y1 - \Delta y\ x1/\Delta x) - 1)$

Solving this we get,

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

Algorithm (m < 1)

Starting coordinates = (X_0, Y_0)

Ending coordinates = (X_n, Y_n)

1. Calculate Δx and Δy from the given input. These parameters are calculated as,

   $\Delta x = X_n - X_0$

   $\Delta y = Y_n - Y_0$

2. Calculate the decision parameter P_k using $P_0 = 2|\Delta y| - |\Delta x|$
3. Suppose the current point is (X_k, Y_k) and the next point is (X_{k+1}, Y_{k+1}). Find the next point depending on the value of decision parameter P_k. Follow the below two cases:

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

   If $P_k < 0$ Then,

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

   $y_{k+1} = y_k$

   else

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

   $y_{k+1} = y_k + 1$
4. Keep repeating Step-03 until the end point is reached or number of iterations equals to $(\Delta x-1)$ times.

2.2.2 Derivation of the decision parameter P_k (m > 1)

If the slope of line is greater than 1 (m>1) then our Y coordinate will always be incremented and we have to choose between x_k or x_{k+1}.

So, our Line equation will be:

$$y_{k+1} = mx+b$$

Solving it for x, we get $x=(y_{k+1} - b)/m$

Now, In this case our d_1 will be the distance between intersection point x and pixel x_k,

$$d_1 = x - x_k = (y_{k+1} - b)/m - x_k$$

Similarly, d_2 will be the distance between intersection point x and pixel x_{k+1}

$$d_2 = x_{k+1} - x$$

$$d_2 = x_{k+1} - (y_{k+1} - b)/m$$

As y is always incremented, so y_{k+1} will always be y_k+1 and x_{k+1} is next x pixel so we can write it as $x=x_{k+1}$.

Now, we calculate the values of $d_1 - d_2$

$$\begin{aligned} d_1 - d_2 &= (y_k + 1 - b)/m - x_k - [x_k + 1 - (y_k + 1 - b)/m] \\ &= (y_k + 1 - b)/m - x_k - x_k - 1 + (y_k + 1 - b)/m \\ &= 2(y_k + 1 - b)/m - 2x_k - 1 \\ &= 2\Delta x(y_k + 1 - b)/ \Delta y - 2x_k - 1 \end{aligned}$$

To remove Δy from denominator, multiply Δy on both sides;

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

Let $P_k = \Delta y(d1 - d2)$, thus introducing the decision variable

$P_k = 2\Delta x\, y_k - 2\Delta y\, x_k + c$ where $c = 2\Delta x\,(1 - b) - \Delta y$ which is constant

Now our next parameter will be

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

Subtracting P_k from P_{k+1}

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

Since, y_{k+1} will always be $y_k + 1$

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

$P_{k+1} = P_k + 2\Delta x - 2dy\,(x_{k+1} - x_k)$

Now, If $P_k < 0$ i.e. d1<d2 (because Δx is considered as absolute) then,

$x_{k+1} = x_k$ (We will choose the nearest x_k pixel)

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

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

Now, If $P_k > 0$ i.e. d1>d2 Then,

$x_{k+1} = x_k + 1$(We will choose the nearest $y_k + 1$ pixel)

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

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

Now, we will use P_k to get initial parameter P_0 using (x1,y1)

$P_0 = 2\Delta x\, y_1 - 2\Delta y\, x_1 + 2\Delta x - 2\Delta x\, b - \Delta y$

By using $y_1 = m\, x_1 + b$

$P_0 = 2\Delta x\, y_1 - 2\Delta y\, x_1 + 2\Delta x - 2\Delta x(y_1 - mx_1) - \Delta y$

$P_0 = 2\Delta x\, y_1 - 2\Delta y\, x_1 + 2\Delta x - 2\Delta x\, y_1 + 2\Delta y\, x_1 - \Delta y$ (using $m = \Delta y/\Delta x$)

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

Algorithm (m > 1)

Starting coordinates = (X_0, Y_0)

Ending coordinates = (X_n, Y_n)

1. Calculate Δx and Δy from the given input. These parameters are calculated as,

   $\Delta x = X_n - X_0$

   $\Delta y = Y_n - Y_0$

2. Calculate the decision parameter P_k using $P_0 = 2|\Delta x| - |\Delta y|$
3. Suppose the current point is (X_k, Y_k) and the next point is (X_{k+1}, Y_{k+1}). Find the next point depending on the value of decision parameter P_k. Follow the below two cases:

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

   If $P_k < 0$ Then,

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

   $x_{k+1} = x_k$

   else

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

   $x_{k+1} = x_k + 1$
4. Keep repeating Step-03 until the end point is reached or number of iterations equals to $(\Delta y-1)$ times.

Advantages:

- It is easy to implement, fast and incremental.
- The points generated by this algorithm are more accurate than DDA Algorithm.
- It uses fixed points only.

Disadvantages:

- Though it improves the accuracy of generated points but still the resulted line is not smooth.
- This algorithm is for the basic line drawing and cannot handle jaggies.

Example: Calculate the points between the coordinates (9, 18) and (14, 22).

Solution:

Starting coordinates = $(X_0, Y_0) = (9, 18)$

Ending coordinates = $(X_n, Y_n) = (14, 22)$

Step-01: Calculate ΔX and ΔY from the given input.

$\Delta X = X_n - X_0 = 14 - 9 = 5$

$\Delta Y = Y_n - Y_0 = 22 - 18 = 4$

Step-02: Calculate the decision parameter.

$P_k = 2|\Delta Y| - |\Delta X| = 2 \times 4 - 5 = 3$

So, decision parameter $P_k = 3$

Step-03: As $P_k >= 0$, so case-02 is satisfied. Thus,

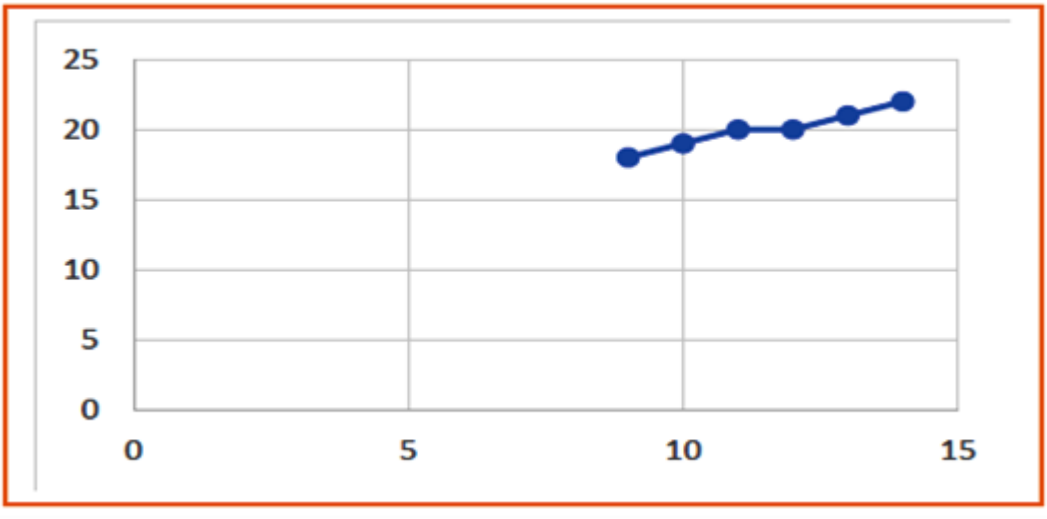
$P_{k+1} = P_k + 2|\Delta Y| - 2|\Delta X| = 3 + (2 \times 4) - (2 \times 5) = 1$

$X_{k+1} = X_k + 1 = 9 + 1 = 10$

$Y_{k+1} = Y_k + 1 = 18 + 1 = 19$

Similarly, Step-03 is executed until the end point is reached or number of iterations equals to 4 times. (Number of iterations = $\Delta X - 1 = 5 - 1 = 4$)

P_{k+1}	X_{k+1}	Y_{k+1}
3	9	18
1	10	19
-1	11	20
7	12	20
5	13	21
3	14	22



2.3 Midpoint Line Drawing Algorithm

The Mid-Point line plotting algorithm follows the bisection method to divide the line into equal partitions. Using this, we can draw a line with close approximation between two points. As usual, we will have to find the next pixel position of X and Y coordinates depending the decision parameter using slope of the line. So, let's consider case for line with slop < 1.

2.3.1 Derivation of the decision parameter P_k (for m<1)

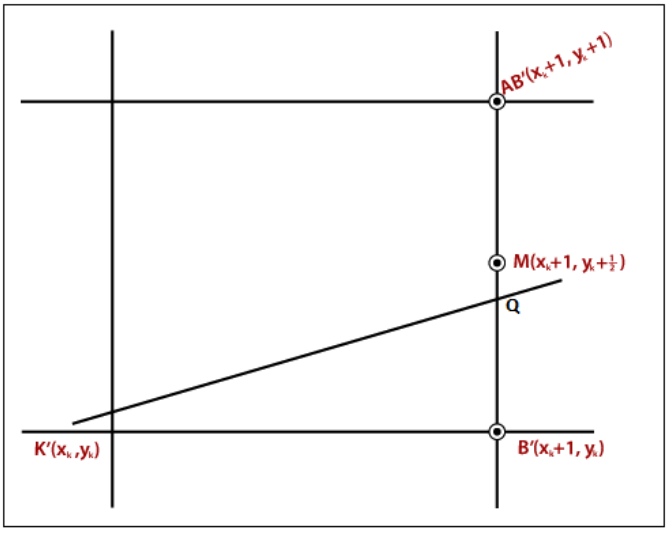
Below are some assumptions to keep algorithm simple:

1. We draw line from left to right.

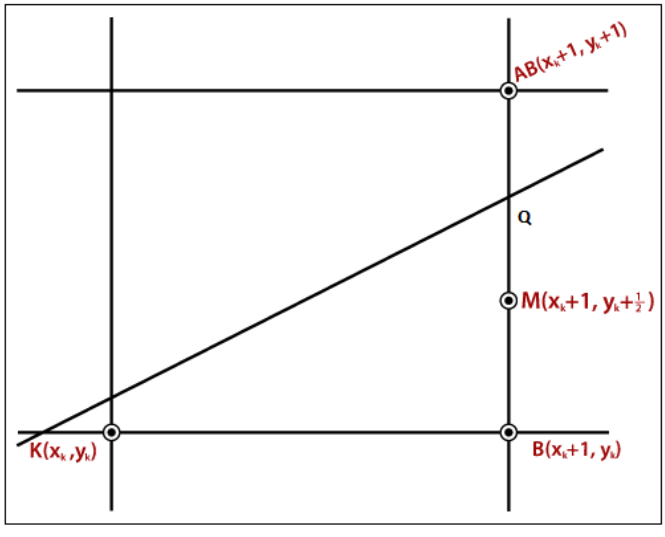
2. x1 < x2 and y1< y2
3. Slope of the line is between 0 and 1. We draw a line from lower left to upper right.

In Mid-Point algorithm we do following:

1. Find middle of two possible next points. Middle of B'($X_k + 1$, Y_k) and AB'($X_k + 1$, $Y_k + 1$) is M(X_{k+1}, $Y_k + (1/2)$).
2. If M is above the line, it means Q is near to B', then choose B' as next point.
3. If M is below the line, it means Q is near to AB', then choose AB' as next point



Case 1 (P_k < 0)



Case 2 (P_k >= 0)

Let us consider a line y = mx + B.

We can re-write the equation as :

$y = (\Delta y/\Delta x)x + B$ or

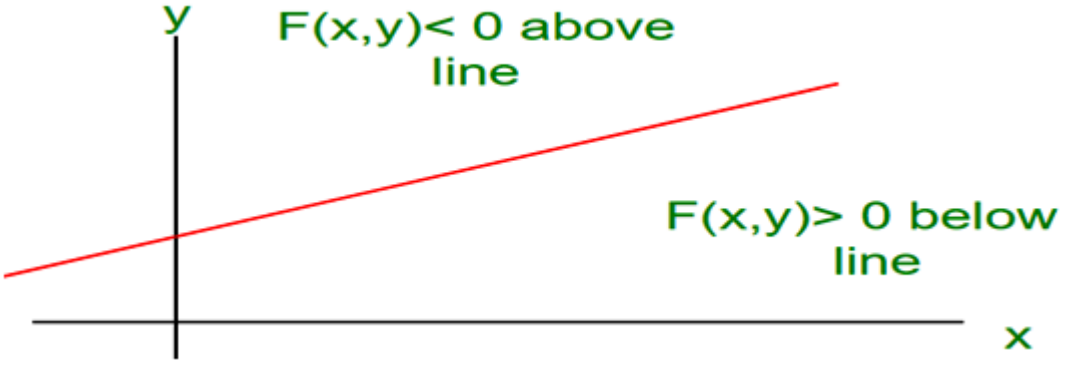
$(\Delta y)x - y(\Delta x) + B\Delta x = 0$

Let $F(x, y) = (\Delta y)x - y(\Delta x) + B\Delta x$ -----(1)

So comparing it with ax + by + c.
$a = \Delta y$
$b = -\Delta x$
$c = B\Delta x$

We are given two end points of a line (under the assumptions we made in the begining)

1. For all points (x,y) on the line, the solution to F(x, y) is 0.
2. For all points (x,y) above the line, F(x, y) result in a negative number i.e. here, if for point M, we are getting < 0 value means the point M is above the line passing through Q. Hence, the point Q is near to B' i.e. y_k.
3. For all points (x,y) below the line, F(x, y) result in a positive number i.e. here, if for point M, we are getting > 0 value means the point M is below the line passing through Q. Hence, the point Q is near to AB i.e. y_{k+1}.



This relationship is used to determine the relative position of M.

$$M = (X_k + 1,\ Y_k + (1/2))$$

Putting these into line equation, our decision parameter P_k is,
$P_k = F(X_k + 1, Y_k + (1/2)) = a(X_k + 1) + b(Y_k + (1/2)) + c$

$P_{k+1} = F(X_{k+1} + 1, Y_{k+1} + (1/2)) = a(X_{k+1} + 1) + b(Y_{k+1} + (1/2)) + c$
$P_{k+1} - P_k = a(X_{k+1} + 1) - a(X_k + 1) + b(Y_{k+1} + (1/2)) - b(Y_k + (1/2)) + c-c$
$P_{k+1} = P_k + a(X_{k+1} + 1- X_k - 1) + b(Y_{k+1} + (1/2)- Y_k - (1/2))$

$= P_k + a(X_{k+1} - X_k)+ b(Y_{k+1} - Y_k)$

but, $X_{k+1}= X_k + 1$ always. Hence,

$P_{k+1} = P_k + a(X_k + 1 - X_k)+ b(Y_{k+1} - Y_k)$

$= P_k + a + b(Y_{k+1} - Y_k)$

Case 1 ($P_k < 0$): It means line of M is above Q line. Hence, equation of M gives < 0 values and Q is near to current Y_k. So Y_k is not incremented and B' is chosen. So, putting $Y_{k+1} = Y_k$

$P_{k+1} = P_k + a + b(Y_k - Y_k)$

$= P_k + a$

$= P_k + \Delta y$ (as $a = \Delta y$, $b = -\Delta x$)

Case 2 ($P_k >= 0$): It means line of M is below Q line. Hence, equation of M gives > 0 values and Q is near to Y_k+1. So Y_k is incremented by 1 and AB' is chosen. So, putting $Y_{k+1} = Y_k + 1$

$P_{k+1} = P_k + a + b(Y_k + 1 - Y_k)$

$= P_k + a + b$

$= P_k + \Delta y - \Delta x$ (as $a = \Delta y$, $b = -\Delta x$)

Calculation For initial value of decision parameter P_0:

$P_0 = F(X1 + 1 , Y1 + (1/2))$
$= a(X1 + 1) + b(Y1 + (1/2)) + c$
$= aX1 + bY1 + c + a + (b/2)$
$= F(X1,Y1) + a + (b/2)$
$= a + (b/2)$ (as $F(X1, Y1) = 0$)
$= \Delta y – (\Delta x/2)$ (as $a = \Delta y$, $b = -\Delta x$)

Algorithm (m < 1)

1. Input (X1,Y1) and (X2,Y2)
2. $\Delta y = Y2 - Y1$ $\Delta x = X2 - X1$
3. Calculate initial value of decision parameter using $P_0 = \Delta y - (\Delta x/2)$

   $x = X1$, $y = Y1$
4. Suppose the current point is (X_k, Y_k) and the next point is (X_{k+1}, Y_{k+1}). Find the next point depending on the value of decision parameter P_k. Follow the below two cases:

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

   if ($P_k < 0$)

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

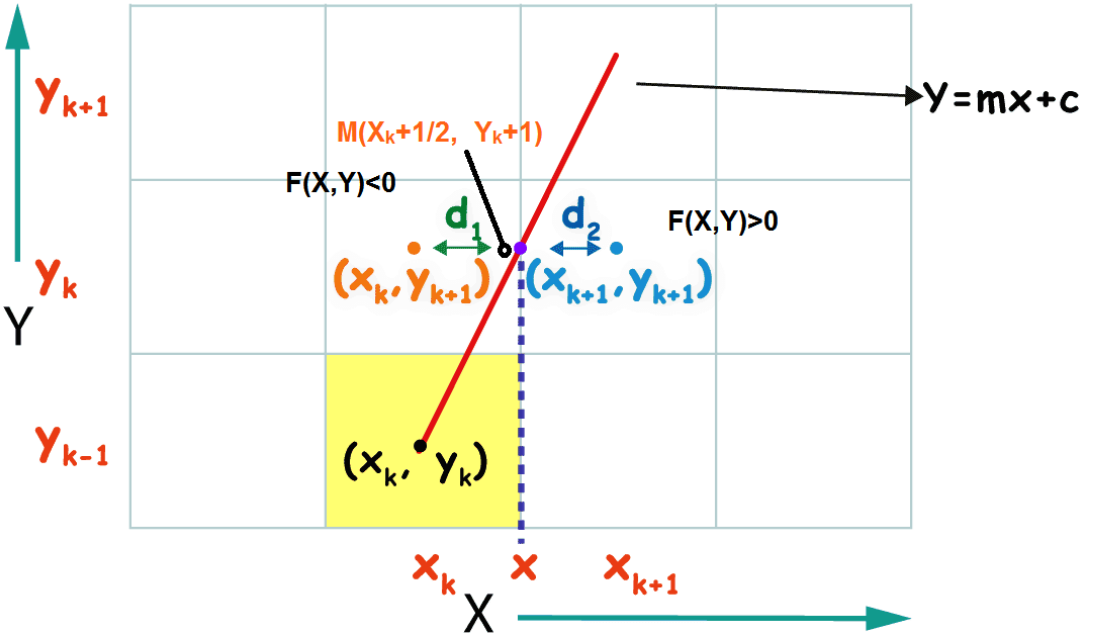
   else

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

   $y_{k+1} = y_k + 1$
5. Repeat step 4 until $x_{k+1} = X2$ and/or $y_{k+1} = Y2$

2.3.2 Derivation of the decision parameter P_k (for m > 1)

This algorithm can be derived using reflection property of line and coordinate.



Let us consider a line y = mx + B.

We can re-write the equation as :

y = (Δy/Δx)x + B or

-(Δy)x + y(Δx) - BΔx = 0

Let F(x, y) = -(Δy)x + y(Δx) - BΔx -----(1)

So comparing it with ax + by + c.
a = -Δy
b = Δx
c = -BΔx

This relationship is used to determine the relative position of M.

$$M = (X_k + (1/2),\ Y_k + 1)$$

Putting these into line equation, our decision parameter P_k is,

$P_k = F(X_k + (1/2), Y_k + 1) = a(X_k + (1/2)) + b(Y_k + 1) + c$

$P_{k+1} = F(X_{k+1} + (1/2), Y_{k+1} + 1) = a(X_{k+1} + (1/2)) + b(Y_{k+1} + 1) + c$

$P_{k+1} - P_k = a(X_{k+1} + (1/2)) - a(X_k + (1/2)) + b(Y_{k+1} + 1) - b(Y_k + 1) + c - c$

$P_{k+1} = P_k + a(X_{k+1} + (1/2) - X_k - (1/2)) + b(Y_{k+1} + 1 - Y_k - 1)$

$= P_k + a(X_{k+1} - X_k) + b(Y_{k+1} - Y_k)$

but, $Y_{k+1} = Y_k + 1$ always. Hence,

$P_{k+1} = P_k + a(X_{k+1} - X_k) + b(Y_k + 1 - Y_k)$

$= P_k + a(X_{k+1} - X_k) + b$

Case 1 ($P_k < 0$): It means Q line is on left side of line of M. Hence, equation of M gives < 0 values and Q is near to current X_k. So X_k is not incremented, putting $X_{k+1} = X_k$

$P_{k+1} = P_k + a(X_k - X_k) + b$

$= P_k + b$

$= P_k + \Delta x$ (as $a = -\Delta y$, $b = \Delta x$)

Case 2 ($P_k >= 0$): It means Q line is on right side of line of M. Hence, equation of M gives > 0 values and Q is near to X_k+1. So $X_{k+1} = X_k + 1$

$P_{k+1} = P_k + a(X_k + 1 - X_k) + b$

$= P_k + a + b$

$= P_k + \Delta x - \Delta y$ (as $a = -\Delta y$, $b = \Delta x$)

Calculation For initial value of decision parameter P_0:

$P_0 = F(X1 + (1/2), Y1 + 1)$

$= a(X1 + (1/2)) + b(Y1 + 1) + c$

$= aX1 + bY1 + c + (a/2) + b$

$= F(X1,Y1) + (a/2) + b$

$= (a/2) + b$ (as $F(X1, Y1) = 0$)

$= \Delta x - (\Delta y/2)$. (as $a = -\Delta y$, $b = \Delta x$)

Algorithm (m > 1)

1. Input (X1,Y1) and (X2,Y2)
2. $\Delta y = Y2 - Y1$ $\Delta x = X2 - X1$
3. Calculate initial value of decision parameter $P_0 = \Delta x - (\Delta y/2)$

   $x = X1$, $y = Y1$
4. Suppose the current point is (X_k, Y_k) and the next point is (X_{k+1}, Y_{k+1}). Find the next point depending on the value of decision parameter P_k. Follow the below two cases.

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

   if ($P_k < 0$)

   $P_{k+1} = P_k + \Delta x$

   else

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

   $x_{k+1} = x_k + 1$
5. Repeat step 4 until $x_{k+1} = X2$ and/or $y_{k+1} = Y2$

Advantages:

- Accuracy of finding points is a key feature of this algorithm.
- It is simple to implement and uses basic arithmetic operations.
- It takes less time for computation.
- The resulted line is smooth as compared to other line drawing algorithms.

Disadvantages:

- This algorithm may not be an ideal choice for complex graphics and images.
- In terms of accuracy of finding points, improvement is still needed.
- There is no any remarkable improvement made by this algorithm.

Example: Calculate the points between the coordinates (20,10) and (30,18).

Solution:

Starting coordinates = $(X_0, Y_0) = (20, 10)$

Ending coordinates = $(X_n, Y_n) = (30, 18)$

Step-01: Calculate ΔX and ΔY from the given input.

$\Delta X = X_n - X_0 = 30 - 20 = 10$

$\Delta Y = Y_n - Y_0 = 18 - 10 = 8$

Step-02: Calculate P_0 as-

$P_0 = \Delta Y - \Delta X/2 = 8 - 10/2 = 3$

Step-03: As $P_0 >= 0$, so case-02 is satisfied.

Thus,

$X_{k+1} = X_k + 1 = 20 + 1 = 21$

$Y_{k+1} = Y_k + 1 = 10 + 1 = 11$

$P_{k+1} = P_k + \Delta Y - \Delta X = 3 + 8 - 10 = 1$

As $P_k >= 0$, so case-02 is satisfied.

$X_{k+1} = X_k + 1 = 21 + 1 = 22$

$Y_{k+1} = Y_k + 1 = 11 + 1 = 12$

$P_{k+1} = P_k + \Delta Y - \Delta X = 1 + 8 - 10 = -1$

As $P_k < 0$, so case-01 is satisfied.

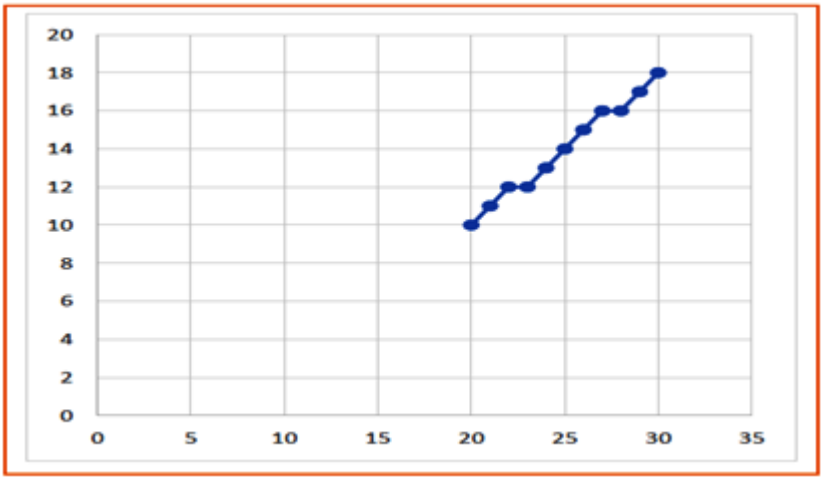
$X_{k+1} = X_k + 1 = 22 + 1 = 23$

$Y_{k+1} = Y_k = 12$

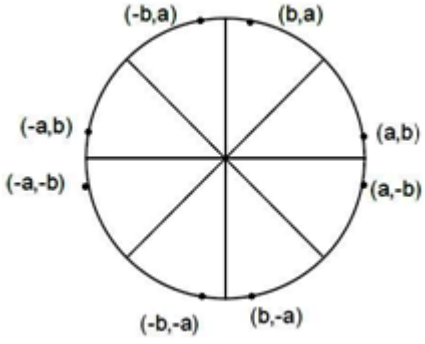
$P_{k+1} = P_k + \Delta Y = -1 + 8 = 7$

Similarly, Step-03 is executed until the end point is reached.

P_{k+1}	X_{k+1}	Y_{k+1}
3	20	10
1	21	11
-1	22	12
7	23	12
5	24	13
3	25	14
1	26	15
-1	27	15
7	28	16
5	29	17
3	30	18



3. CIRCLE



A circle is defined as a set of points that are the same distance from a common point. The common point is known as the centre and the distance from the centre of the circle to any point on its circumference is called the radius. It is an eight-way symmetric figure which can be divided into four quadrants and each quadrant has two octants. This symmetry helps in drawing a circle on a computer by knowing points of any one octant.

There are two methods to define a circle in computer graphics:

1. Direct or Polynomial Method: In this method, a circle is defined with the help of a polynomial equation i.e.

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

Where, (x_c, y_c) are coordinates of circle and r is radius of circle.

For each value of x, value of y can be calculated using,

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

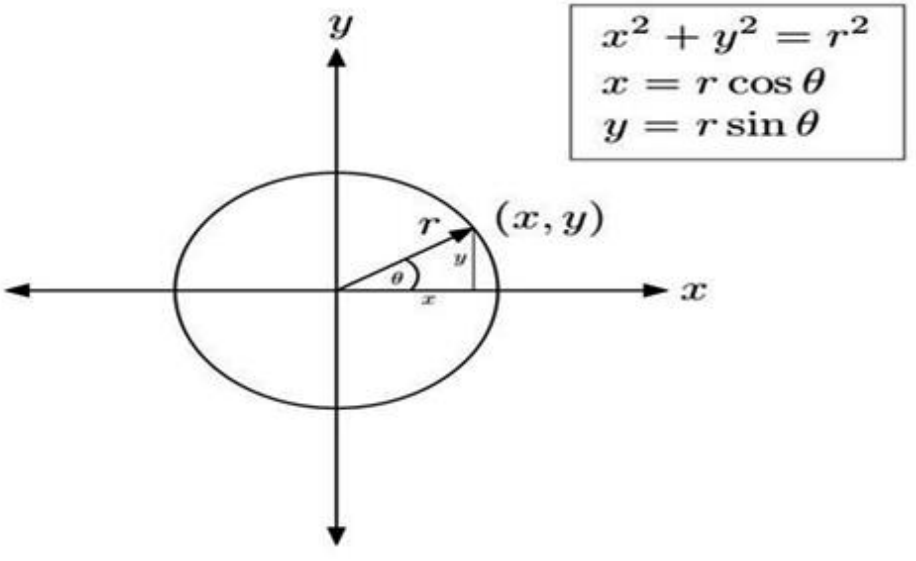
This is a very ineffective method because for each point value of x_c, x and r are squared and then subtracted and then the square root is calculated, which leads to high time complexity.

2. Polar coordinates Method: In this method, the coordinates are converted into polar coordinates. The coordinates are thus given by:

$$x = r\cos\theta$$

$$y = r\sin\theta$$

where r = radius and θ = current angle

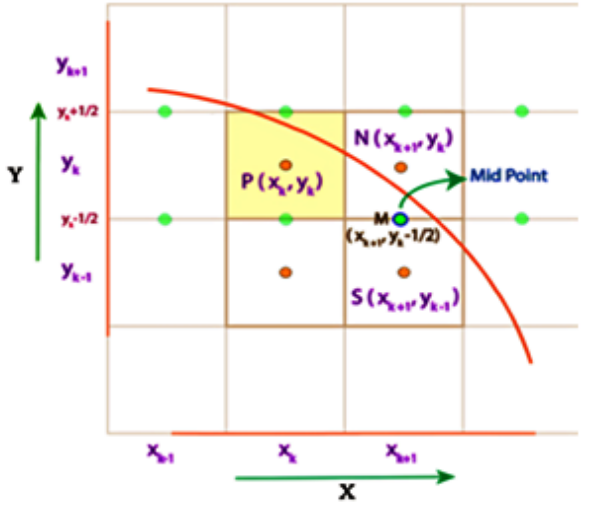


For the first octant i.e. from $\theta = 0^0$ to $\theta = 45^0$, the values of x and y are calculated using above equations. Simultaneously, all the points for the rest of the octants can be calculated using the eight-way symmetry property of the circle. Thus, the plotting of the circle is carried out.

3.1 Midpoint Circle Algorithm

The mid-point circle drawing algorithm is used to calculate all the perimeter points of a circle. Similar to mid-point line algorithm, this algorithm also calculates the mid-point between the two pixels which helps in calculating the decision parameter. The value of the decision parameter will decide which pixel should be chosen for drawing the circle. This algorithm only calculates the points for one octant and the points for other octants are generated using the eight-way symmetry for the circle. The algorithm works in the following way: Suppose a mid-point with coordinates (x', y') is put in the general equation of the circle $x^2 + y^2 - r^2$ gives following three values:

- If it is 0 then the given point lies on the circle boundary and any pixel can be chosen.
- If it is < 0 then the given point lies inside the circle boundary and the upper pixel can be chosen.
- If it is > 0 then the given point lies outside the circle boundary and the lower pixel is chosen.



Derivation of the decision parameter P_k

Let there be two pixels: One pixel which is outside (N), and the other pixel which is inside (S) the circle boundary,

Let their coordinates be $N(x_k + 1, y_k)$ and $S(x_k + 1, y_k - 1)$

Then, the co-ordinates of mid-point be

$M_P = [((x_k + 1 + x_k + 1)/2) , ((y_k + y_k - 1)/2)]$

$= (x_k + 1 , y_k - (1/2))$

Now, put M_P in the equation of circle $x^2 + y^2 - r^2 = 0$

$$(x_k + 1)^2 + (y_k - (1/2))^2 - r^2$$

Using the mid-point as the decision parameter,

$P_k = (x_k + 1)^2 + (y_k - (1/2))^2 - r^2$ -----(1)

Let us define the successive decision parameter, P_{k+1}:

$P_{k+1} = (x_{k+1} + 1)^2 + (y_{k+1} - (1/2))^2 - r^2$ -----(2)

Subtracting eq.(1) from eq.(2);

$P_{k+1} - P_k = (x_{k+1} + 1)^2 + (y_{k+1} - (1/2))^2 - r^2 - [(x_k + 1)^2 + (y_k - (1/2))^2 - r^2]$

Now put $x_{k+1} = x_k + 1$

$P_{k+1} - P_k = (x_{k+1} + 1)^2 + (y_{k+1} - (1/2))^2 - r^2 - [(x_k + 1)^2 + (y_k - (1/2))^2 - r^2]$

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

$= (x_k)^2 + 4 x_k + 4 - (x_k)^2 - 2x_k - 1 + (y_{k+1})^2 - y_{k+1} + (1/4) - (y_k)^2 + y_k - (1/4)$

$P_{k+1} = P_k + 2x_k + 3 + (y_{k+1})^2 - (y_k)^2 - (y_{k+1} - y_k)$

If $P_k < 0$, it means Mid point is inside circle and y_k coordinate is closer to circle boundary. Hence, $y_{k+1} = y_k$ (choose point N)

$$P_{k+1} = P_k + 2x_k + 3$$

If $P_k >= 0$, it means Mid point is outside circle and $y_k - 1$ coordinate is closer to circle boundary. Hence, $y_{k+1} = y_k - 1$ (choose point S)

$$P_{k+1} = P_k + 2x_k - 2y_k + 5$$

Let us calculate the initial decision parameter (P_0) where the initial points will be defined as (0, r) [which is the first point to be plotted of the first octant].

On putting these coordinates in eq. (1) in place of x_k and y_k, we get:

$$P_0 = (0 + 1)^2 + (r - (1/2))^2 - r^2$$

$P_0 = 5/4 - r$ which can be taken as $1 - r$ for Integer arithmetic

Algorithm

Centre point of Circle = (X_c, Y_c)

Radius of Circle = r

1. Assign the starting point coordinates (X_0, Y_0) as $X_0 = 0$ $Y_0 = r$
2. Calculate the value for initial decision parameter $P_0 = 1 - r$
3. Suppose the current point is (X_k, Y_k) and the next point is (X_{k+1}, Y_{k+1}). Find the next point of the first octant depending on the value of decision parameter P_k. Follow the below two cases:

   If $P_k < 0$

$y_{k+1} = y_k$

$P_{k+1} = P_k + 2x_k + 3$

If $P_k > 0$

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

$P_{k+1} = P_k + 2x_k - 2y_k + 5$

4. If the given centre point (X_c, Y_c) is not (0, 0), then do the following and plot the point-

   $X_{plot} = X_c + X_{k+1}$

   $Y_{plot} = Y_c + Y_{k+1}$

5. Keep repeating Step-03 and Step-04 until $X_{plot} < Y_{plot}$.
6. Step-05 generates all the points for one octant.

To find the points for other seven octants, follow the eight symmetry property of circle.

Advantages:

- It is a powerful and efficient algorithm.
- The entire algorithm is based on the simple equation of circle $X^2 + Y^2 = r^2$.
- It is easy to implement from the programmer's perspective.
- This algorithm is used to generate curves on raster displays.

Disadvantages:

- Accuracy of the generating points is an issue in this algorithm.
- The circle generated by this algorithm is not smooth.
- The time consumption of this algorithm is high.

Example 1: Given the centre point coordinates (0, 0) and radius 10, generate all the points to form a circle.

Solution:

Centre Coordinates of Circle $(X_0, Y_0) = (0, 0)$

Radius of Circle = 10

Step-01:

Assign the starting point coordinates (X_0, Y_0) as-

$X_0 = 0$

$Y_0 = r = 10$

Step-02:

Calculate the value of initial decision parameter P_0 as-

$P_0 = 1 – r$

$P_0 = 1 – 10 = -9$

Step-03:

As $P_{initial} < 0$, so case-01 is satisfied.

Thus,

$X_{k+1} = X_k + 1 = 0 + 1 = 1$

$Y_{k+1} = Y_k = 10$

$P_{k+1} = P_k + 2 \times X_{k+1} + 3 = -9 + (2 \times 1) + 3 = -4$

Step-04: This step is not applicable here as the given centre point coordinates is (0, 0).

Step-05: Step-03 is executed similarly until $X_{k+1} >= Y_{k+1}$ as follows:

P_{k+1}	(X_{k+1}, Y_{k+1})
	(0, 10)
-9	(1, 10)
-4	(2, 10)
3	(3, 9)
-4	(4, 8)
7	(5, 7)
8	(6, 6)

Algorithm calculates all the points of octant-1 and terminates. Now, the points of octant-2 are obtained using the mirror effect by swapping X and Y coordinates.

Octant-1 Points	Octant-2 Points
(0, 10)	(6, 6)
(1, 10)	(7, 5)
(2, 10)	(8, 4)
(3, 9)	(9, 3)
(4, 8)	(10, 2)
(5, 7)	(10, 1)
(6, 6)	(10, 0)

Now, the points for rest of the part are generated by following the signs of other quadrants. The other points can also be generated by calculating each octant separately. Here, all the points have been generated with respect to quadrant-1.

Quadrant-1 (X,Y)	Quadrant-2 (-X,Y)	Quadrant-3 (-X,-Y)	Quadrant-4 (X,-Y)
(0, 10)	(0, 10)	(0, -10)	(0, -10)
(1, 10)	(-1, 10)	(-1, -10)	(1, -10)
(2, 10)	(-2, 10)	(-2, -10)	(2, -10)
(3, 9)	(-3, 9)	(-3, -9)	(3, -9)
(4, 8)	(-4, 8)	(-4, -8)	(4, -8)
(5, 7)	(-5, 7)	(-5, -7)	(5, -7)
(6, 6)	(-6, 6)	(-6, -6)	(6, -6)
(6, 6)	(-6, 6)	(-6, -6)	(6, -6)
(7, 5)	(-7, 5)	(-7, -5)	(7, -5)
(8, 4)	(-8, 4)	(-8, -4)	(8, -4)
(9, 3)	(-9, 3)	(-9, -3)	(9, -3)
(10, 2)	(-10, 2)	(-10, -2)	(10, -2)
(10, 1)	(-10, 1)	(-10, -1)	(10, -1)
(10, 0)	(-10, 0)	(-10, 0)	(10, 0)

Example 2: Given the centre point coordinates (4, 4) and radius 10, generate all the points to form a circle.

Solution:

Centre Coordinates of Circle $(X_0, Y_0) = (4, 4)$

Radius of Circle = 10

As stated in the algorithm, We first calculate the points assuming the centre coordinates is (0, 0). At the end, we translate the circle. Step-01, Step-02 and Step-03 are already completed in Problem-01. Now, we find the values of X_{plot} and Y_{plot} using the formula given in Step-04 of the main algorithm.

The following table shows the generation of points for Quadrant-1-

$X_{plot} = X_c + X_{k+1} = 4 + X_{k+1}$

$Y_{plot} = Y_c + Y_{k+1} = 4 + Y_{k+1}$

(X_{k+1}, Y_{k+1})	(X_{plot}, Y_{plot})
(0, 10)	(4, 14)
(1, 10)	(5, 14)
(2, 10)	(6, 14)
(3, 9)	(7, 13)
(4, 9)	(8, 13)
(5, 7)	(9, 11)
(6, 6)	(10, 10)

(7, 5)	(11, 9)
(8, 6)	(12, 10)
(8, 4)	(12, 8)
(9, 3)	(13, 7)
(10, 2)	(14, 6)
(10, 1)	(14, 5)
(10, 0)	(14, 4)

The following table shows the points for all the quadrants:

Quadrant-1 (X,Y)	Quadrant-2 (-X,Y)	Quadrant-3 (-X,-Y)	Quadrant-4 (X,-Y)
(4, 14)	(-4, 14)	(-4, -14)	(4, 14)
(5, 14)	(-5, 14)	(-5, -14)	(5, 14)
(6, 14)	(-6, 14)	(-6, -14)	(6, 14)
(7, 13)	(-7, 13)	(-7, -13)	(7, 13)
(8, 13)	(-8, 13)	(-8, -13)	(8, 13)
(9, 11)	(-9, 11)	(-9, -11)	(9, 11)
(10, 10)	(-10, 10)	(-10, -10)	(10, 10)
(11, 9)	(-11, 9)	(-11, -9)	(11, 9)
(12, 10)	(-12, 10)	(-12, -10)	(12, 10)
(12, 8)	(-12, 8)	(-12, -8)	(12, 8)
(13, 7)	(-13, 7)	(-13, -7)	(13, 7)
(14, 6)	(-14, 6)	(-14, -6)	(14, 6)
(14, 5)	(-14, 5)	(-14, -5)	(14, 5)
(14, 4)	(-14, 4)	(-14, -4)	(14, 4)

4. ELLIPSE

Ellipse is defined as the geometric figure which is the set of all points on a plane whose distance from two fixed points known as the foci remains constant. It consists of two axes: major and minor axes where the major axis is the longest diameter and minor axis is the shortest diameter. Unlike circle, the ellipse has four-way symmetry property which means that only the quadrants are symmetric while the octants are not. Here, we will calculate the points for one quadrant while the points for the remaining three can be calculated using the symmetry of points.

4.1 Midpoint Ellipse Algorithm

In computer graphics, the mid-point ellipse algorithm is an incremental method of drawing an ellipse. It is very similar to the mid-point algorithm used in the generation of a circle. The mid-point ellipse drawing algorithm is used to calculate all the perimeter points of an

ellipse. In this algorithm, the mid-point between the two pixels is calculated which helps in calculating the decision parameter. The value of the decision parameter determines whether the mid-point lies inside, outside, or on the ellipse boundary and the then position of the mid-point helps in drawing the ellipse.

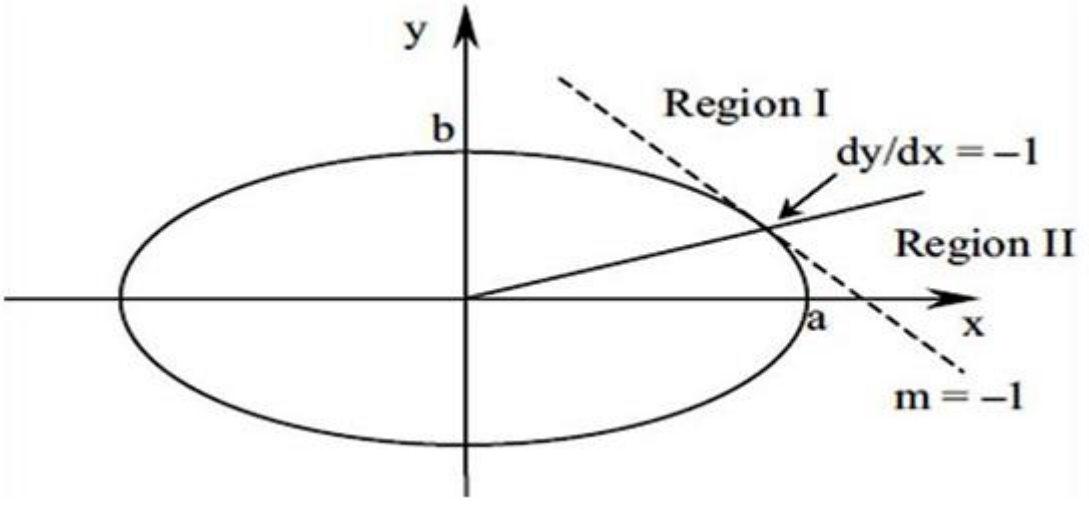
$$r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2 = 0 \qquad \text{-----(1)}$$

where, r_y : semi-minor axis and r_x : semi-major axis

In the ellipse each quadrant is divided into two regions, R1 and R2 respectively. We know that the slope of an ellipse is given by:

$$m = dy / dx = - (2 r_y^2 x / 2 r_x^2 y) = -1$$

The region R1 has the value of slope as $m < -1$ while R2 has the value of slope as $m > -1$. The starting point for R1 will be $(0, r_y)$ and for R2, the ending points of R1 will become the starting points of R2 when the slope becomes greater than -1. That's why we need to keep on checking the value of slope while plotting the points for R1 to know whether we have reached R2 or not.



We will do the derivation for each region:

Derivation of the decision parameter $P1_k$ for Region-1

Let us consider two pixels: one which is outside (A) and the other which is inside (B) respectively.

Let their coordinates be $A(x_k + 1, y_k)$ and $B(x_k + 1, y_k - 1)$

Value for their mid-point will be:

$M_p = [((x_k + 1 + x_k + 1)/2) ,((y_k + y_k - 1)/2)]$

$= (x_k + 1 , y_k - (1/2))$

Putting M_p in eqn (1):

$r_y^2 (x_k + 1)^2 + r_x^2 (y_k - (1/2))^2 - r_x^2 r_y^2$

Let us define this statement as our decision variable:

$P1_k = r_y^2 (x_k + 1)^2 + r_x^2 (y_k - (1/2))^2 - r_x^2 r_y^2$ -----(2)

Successive parameter can be defined as:

$P1_{k+1} = r_y^2 (x_{k+1} + 1)^2 + r_x^2 (y_{k+1} - (1/2))^2 - r_x^2 r_y^2$ -----(3)

Now, subtract eqn (2) from eqn (3);

$P1_{k+1} - P1_k = [r_y^2 (x_{k+1} + 1)^2 + r_x^2 (y_{k+1} - (1/2))^2 - r_x^2 r_y^2] - [r_y^2 (x_k + 1)^2 + r_x^2 (y_k - (1/2))^2 - r_x^2 r_y^2]$

$= r_y^2 [(x_{k+1} + 1)^2 - (x_k + 1)^2] + r_x^2 [(y_{k+1} - (1/2))^2 - (y_k - (1/2))^2]$

As the x-coordinate remains same in both the pixels, put $x_{k+1} = x_k + 1$

$P1_{k+1} - P1_k = r_y^2 [(x_k + 1 + 1)^2 - (x_k + 1)^2] + r_x^2 [(y_{k+1} - (1/2))^2 - (y_k - (1/2))^2]$

$= r_y^2 [2x_k + 3] + r_x^2 [(y_{k+1})^2 - y_{k+1} - (y_k)^2 + y_k]$

$P1_{k+1} = P1_k + r_y^2 [2x_k + 3] + r_x^2 [(y_{k+1})^2 - y_{k+1} - (y_k)^2 + y_k]$

If $P1_k < 0$, it means Mid point is inside ellipse and y_k coordinate is closer to circle boundary. Hence, $y_{k+1} = y_k$ (Choose point A)

$$P1_{k+1} = P1_k + r_y^2 (3 + 2x_k)$$

If $P1_k >= 0$, it means Mid point is outside ellipse and $y_k - 1$ coordinate is closer to circle boundary. Hence, $y_{k+1} = y_k - 1$ (Choose point B)

$$P1_{k+1} = P1_k + r_y^2 (3 + 2x_k) + 2r_x^2 (1 - y_k)$$

Initial Decision Parameter

Put initial point$(0, r_y)$ in eq.(2)

$P1_0 = r_y^2 (0 + 1)^2 + r_x^2 (r_y - (1/2))^2 - r_x^2 r_y^2$

$$P1_0 = r_y^2 + (r_x^2/4) - r_x^2 r_y$$

Derivation of the decision parameter $P2_k$ for Region-2

Let us consider two pixels: one which is outside(P) and the other which is inside(Q) respectively.

Let their coordinates be $P(x_k + 1, y_k - 1)$ and $Q(x_k, y_k - 1)$

Value for their mid-point will be:

$M_p = [((x_k + 1 + x_k)/2), ((y_k - 1 + y_k - 1)/2)]$

$= (x_k + (1/2), y_k - 1)$

Putting M_p in eq.(1):

$r_y^2 (x_k + (1/2))^2 + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2$

Let us define this statement as our decision variable:

$P2_k = r_y^2 (x_k + (1/2))^2 + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2$ ----(4)

Successive parameter can be defined as:

$P2_{k+1} = r_y^2 (x_{k+1} + (1/2))^2 + r_x^2 (y_{k+1} - 1)^2 - r_x^2 r_y^2$ ----(5)

Now, subtract eq.(4) from eq.(5);

$P2_{k+1} - P2_k = [r_y^2 (x_{k+1} + (1/2))^2 + r_x^2 (y_{k+1} - 1)^2 - r_x^2 r_y^2] - [r_y^2 (x_k + (1/2))^2 + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2]$

$= r_y^2 [(x_{k+1} + (1/2))^2 - (x_k + (1/2))^2] + r_x^2 [(y_{k+1} - 1)^2 - (y_k - 1)^2]$

As the y-coordinate remains same in both the pixels, put $y_{k+1} = y_k - 1$

By simplifying the equation we get,

$P2_{k+1} - P2_k = r_y^2 [(x_{k+1})^2 + x_{k+1} - (x_k)^2 - x_k] + r_x^2 [3 - 2y_k]$

$P2_{k+1} = P2_k + r_y^2 [(x_{k+1})^2 + x_{k+1} - (x_k)^2 - x_k] + r_x^2 [3 - 2y_k]$

If $P2_k <= 0$, it means Mid point is inside ellipse and $x_k + 1$ coordinate is closer to circle boundary. Hence, $x_{k+1} = x_k + 1$ (Choose point P)

$$P2_{k+1} = P2_k + 2r_y^2(1 + x_k) + r_x^2(3 - 2y_k)$$

If $P2_k > 0$, it means Mid point is outside ellipse and x_k coordinate is closer to circle boundary. Hence, $x_{k+1} = x_k$ (Choose point Q)

$$P2_{k+1} = P2_k + r_x^2(3 - 2y_k)$$

Initial Decision Parameter:

Putting the ending point of region R1

Where, m > -1 i.e. (x, y) in eq.(4)

$$P2_0 = r_y^2 (x + (1/2))^2 + r_x^2 (y-1)^2 - r_x^2 r_y^2$$

The algorithm containing both region formula is given on the next page:

Algorithm

1. Declare r_x, r_y, x, y, m, dx, dy, $P1_k$, $P2_k$.
2. Initialize initial point of region1 as x=0, $y = r_y$
3. Calculate $P1_0 = r_y^2 + r_x^2 / 4 - r_x^2 r_y$

   $dx = 2 r_y^2 x$

   $dy = 2 r_x^2 y$
4. Update values of dx and dy after each iteration.
5. Repeat steps while ($2r_y^2 x < 2r_x^2 y$) (or dx < dy):

   $x_k = x_k + 1$;

   if($P1_k < 0$)

   $P1_{k+1} = P1_k + r_y^2(3 + 2x_k)$

   else

   $P1_{k+1} = P1_k + r_y^2(3 + 2x_k) + 2r_x^2(1 - y_k)$

   $y_k = y_k - 1$
6. When $2r_y^2 x >= 2r_x^2 y$ (or dx ≥ dy), plot region 2:
7. Calculate $P2_0 = r_y^2(x + (1/2))^2 + r_x^2(y - 1)^2 - r_x^2 r_y^2$
8. Repeat till ($y_k > 0$)

   $y_k = y_k - 1$

   If ($P2_k > 0$)

   $P2_{k+1} = P2_k + r_x^2(3 - 2y_k)$

   else

   $x_k = x_k + 1$

   $P2_{k+1} = P2_k + 2r_y^2(1 + x_k) + r_x^2(3 - 2y_k)$

Example: Given the centre point coordinates (0, 0) and major axis r_x = 4 and minor axis r_y = 2, generate all the points to form a circle.

Solution:

Centre Coordinates of Circle (X_0, Y_0) = (0, 0)

Major Axis r_x= 4

Minor Axis r_y= 2

Starting with (0,2)

Calculate $P1_0 = r_y^2 + (r_x^2 / 4) - r_x^2 r_y$

$= 4 - 16(2) + 16/4 = -24$

$P1_{k+1}$	X_{k+1}	Y_{k+1}	$r_y^2 x$	$r_x^2 y$
	0	2	4(0) = 0	16(2) = 32
-24	1	2	4	32
-4	2	2	8	32
24	3	1	12	16
60	4	0	16	0

Here, $r_y^2 x > r_x^2 y$. So R1 ends.

Now, to calculate $P2_0 = r_y^2 (x + (1/2))^2 + r_x^2 (y-1)^2 - r_x^2 r_y^2$, we will have to use the last coordinates which were part of region R1, here (3,1). Putting them into $P2_0$, we get $P2_0$=-15

$P2_{k+1}$	X_{k+1}	Y_{k+1}
	3	1
-15	4	0

As we have reached to Y_{k+1} to 0, means we have reached to (r_x, 0). So region R2 ends here. Finally points for all the four quadrants are as under:

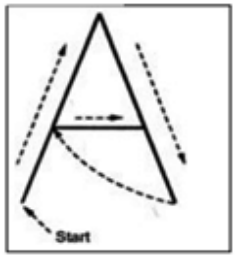
Quadrant-1 (X,Y)	Quadrant-2 (-X,Y)	Quadrant-3 (-X,-Y)	Quadrant-4 (X,-Y)
(0, 2)	(0, 2)	(0, -2)	(0, -2)
(1, 2)	(-1, 2)	(-1, -2)	(1, -2)
(2, 2)	(-2, 2)	(-2, -2)	(2, -2)
(3, 1)	(-3, 1)	(-3, -1)	(3, -1)
(4, 0)	(-4, 0)	(-4, 0)	(4, 0)

5. CHARACTER GENERATION

The design style for a set of characters is called a Typeface or Font. There are three methods of character generation.

5.1 Stroke Method

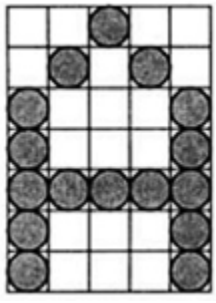
Stroke method is based on natural method of text written by human being. In this method fonts are generated in the form of line, circles and ellipse.



Line drawing algorithm DDA follows this method for line drawing. This method uses small line segments to generate a character. The small series of line segments are drawn like a stroke of pen to form a character. We can build our own stroke method character generator by calls to the line drawing algorithm. Here it is necessary to decide which line segments are needed for each character and then drawing these segments using line drawing algorithm.

5.2 Bitmap Method

Bitmap method is a called dot-matrix method as the name suggests this method uses array of bits for generating a character. These dots are the points for array whose size is fixed. In bit matrix method, when the dots are stored in the form of array, the value 1 in array represent the characters. It means where the dots appear we represent that position with numerical value 1 and the value where dots are not present is represented by 0 in array. It is also called dot matrix because in this method characters are represented by an array of dots in the matrix form. It is a two dimensional array having columns and rows. A 5x7 array is commonly used to represent characters. However 7x9 and 9x13 arrays are also used. Higher resolution devices such as inkjet printer or laser printer may use character arrays that are over 100x100.

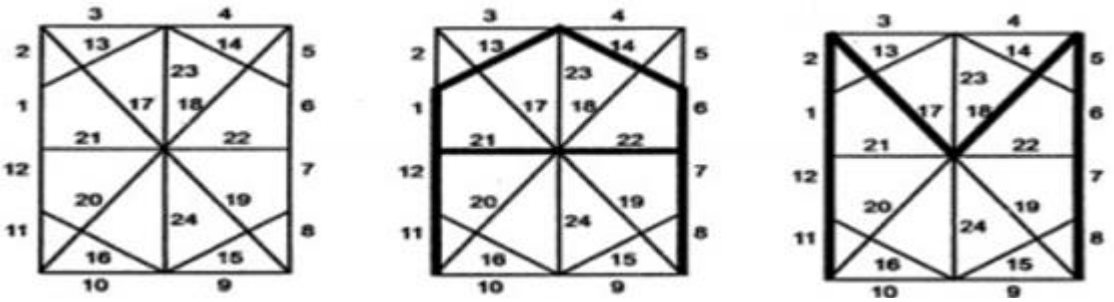


5.3 Starbust Method

In this method a fix pattern of line segments are used to generate characters. Out of these 24 line segments, segments which are required to be displayed for particular character are highlighted. This method of character generation is called starbust method because of its characteristic appearance.

The starbust patterns for characters A and M is given below. The patterns for particular characters are stored in the form of 24 bit code, each bit representing one line segment. The bit is set to one to highlight the line segment; otherwise it is set to zero. For example, 24-bit code for Character A is 1000 0111 0011 1100 0000 1100 and for character M is 1100 1111 0011 0000 1100 0000.

This method of character generation has some disadvantages. As the 24-bits are required to represent a character, more memory is required. It requires code conversion software to display character from its 24- bitcode. Also, character quality is poor and is worst for curve shaped characters.



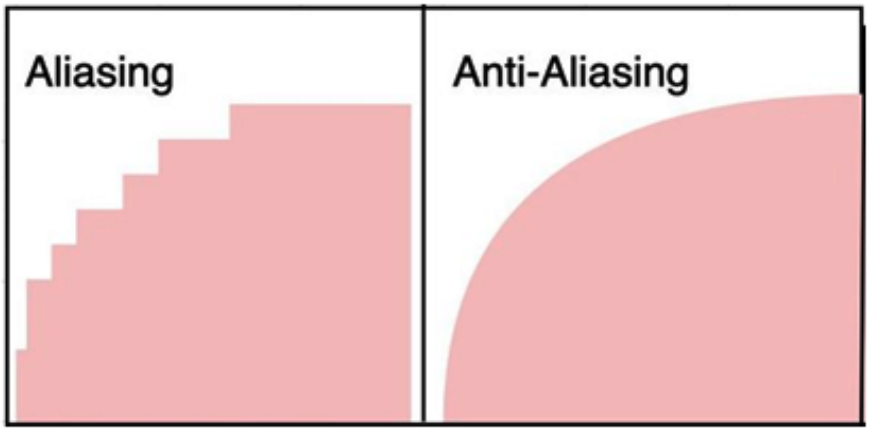
Starburst method with 24 line segments generating characters A and M

6. ANTI ALIASING TECHNIQUES

In the line drawing algorithms, we have seen that all rasterized locations do not match with the true coordinates and we have to select the optimum raster locations to represent a straight line. This problem is severe in low resolution screens. In such screens, line or object appears like a stair-step, as shown in the figure below. Such stair-steps are called jaggies. They do not give a smooth line or curve or object. This effect of getting jaggies, due to alias pixels, is known as aliasing. It is dominant for lines having gentle and sharp slopes. Anti-aliasing is a technique used by users to get rid of jaggies that form on the screen.

Jagged lines

Anti-aliasing tries to smooth out the shape and produce perfect rounded and smooth edges.

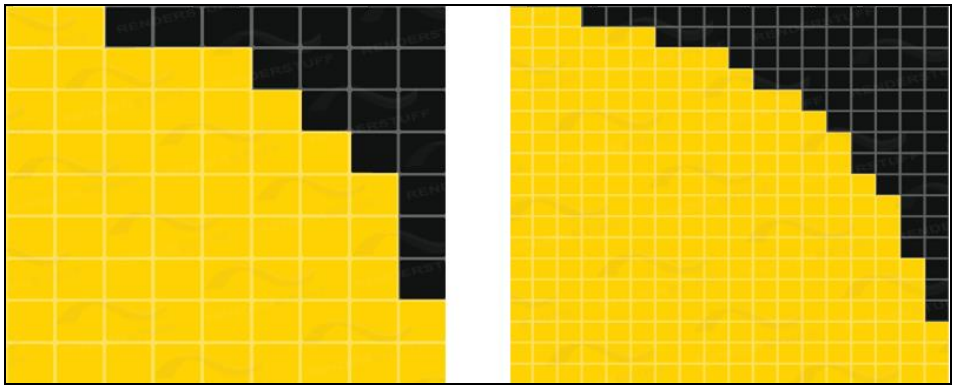


Anti-Aliasing techniques were developed to combat the effects of aliasing. There are many anti-aliasing algorithms divided into following two broad categories.

6.1 Spatial anti-aliasing

To understand how spatial anti-aliasing works, you need to know a couple things about display resolution. Display resolution refers to the number of pixels your monitor uses to create an image. A good monitor should have a display resolution of at least 1920 x 1080 (1900 pixels on the horizontal axis and 1080 pixels on the vertical axis), but lots of monitors these days have resolutions that are even higher. Higher resolutions yield better images because they utilize more pixels. With more pixels, we get a larger variety of colors in the image, and more colors means more detail in the pixel world i.e. image.

Now let's understand how spatial anti-alising works. You have an image at a lower resolution that's full of jaggies. The image is rendered at a higher resolution. At the high resolution, color samples are taken of the excess pixels (new pixels that weren't present in the low-resolution image). The high-resolution image is shrunk down to the original resolution, and each pixel receives a new color that's averaged from the sampled pixels. Essentially, a low-resolution image gets the color accuracy of a high-resolution image. The new colors help the pixels blend together better and the jaggies become less visible.



Following are the spatial anti-aliasing techniques.

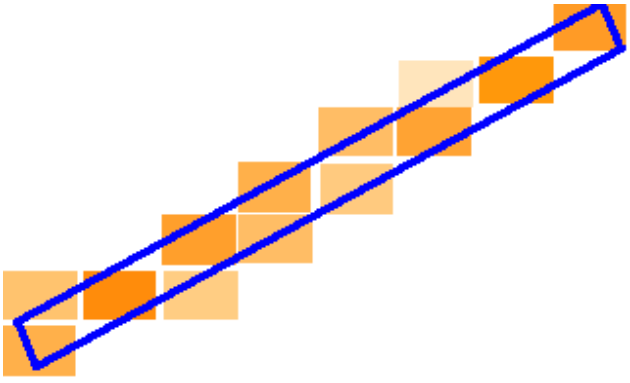
Supersampling anti-aliasing (SSAA) is one of the most effective spatial methods. It's also known as "full-scene anti-aliasing" (FSAA). When a GPU is rendering an image on your display, it makes a distinction between two different objects: a polygon and a texture. First, the GPU draws the polygon: the general shape or outline of an in-game object. Then, the GPU fills in the polygon with a texture

In multisample anti-aliasing (MSAA), only the edges of the polygons are smoothed out - MSAA does not smooth out the textures. This cuts down on processing power a little bit (but not a lot). MSAA is a popular anti-aliasing method among PC gamers. It's effective, but keep in mind that you might still get pixelated textures.

NVIDIA and AMD developed a new type of spatial anti-aliasing method. CSAA (coverage sampling anti-aliasing) was created by NVIDIA, while EQAA (enhanced quality anti-aliasing) was created by AMD. They have different names, but they each work in roughly the same way.

In area sampling, the image can be calculated by considering the intensities over a particular region. Prefiltering methods treat a pixel as an area, and compute pixel colors based on the overlap of the scene's objects with a pixel's area. These techniques compute the shades of gray based on how much of a pixel's area is covered by a object.

For example, a modification to Bresenham's algorithm was developed by Pitteway and Watkinson. In this algorithm, each pixel is given an intensity depending on the area of overlap of the pixels and the line. So, due to the blurring effect along the line edges, the effect of antialiasing is not very prominent, although it still exists. Prefiltering thus amounts to sampling the shape of the object very densely within a pixel region. For shapes other than polygons, this can be very computationally intensive.



0	0	0	1/8	0
0	0	1/4	.914	1/8
0	1/4	.914	1/4	0
1/8	.914	1/4	0	0
0	1/8	0	0	0

Area sampling

Prefiltering

6.2 Post-process anti-aliasing (Supersampling Technique)

In post-process anti-aliasing, each pixel is slightly blurred after it's rendered. The GPU determines where the edge of a polygon is by comparing the color contrast between each two pixels. Two similar pixels indicate that they're part of the same polygon. The pixels are blurred in proportion to their contrast.

Blurring is an effective method for anti-aliasing because it eliminates the stark contrast between awkwardly aligned pixels that are causing jaggies. The downside of the post-process method is that it might make your images appear too blurry. The blurriness is more noticeable in games that have very detailed textures and dynamic lighting features. But

what's great about post-process anti-aliasing is that it's very fast and requires much less processing power than the spatial method.

NVIDIA developed MLAA (morphological anti-aliasing) and AMD developed FXAA (fast approximate anti-aliasing) post-processing methods that work in a very similar fashion. Both methods work in the same way as described above.

Temporal anti-aliasing (TXAA) is a film style technique that seeks to maintain a very smooth level of motion as you move through a virtual environment. TXAA is a very complex method that utilizes both supersampling and blurring to create sharp graphics and graceful motion. Another method, Subpixel morphological anti-aliasing (SMAA) is widely used in the PC gaming world.

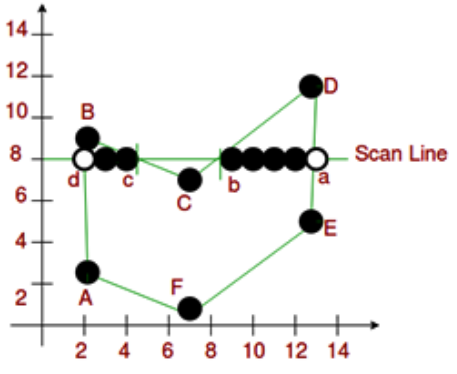
7. AREA FILLING (POLYGON FILLING ALGORITHM)

Figures on a computer screen can be drawn using polygons and polygons can be created using any of the line drawing algorithms. For filling polygons with particular colors, you need to determine the pixels falling on the border of the polygon and those which fall inside the polygon. To fill these figures with color, there are two famous algorithms for this purpose: Scanline fill and Boundary fill algorithms.

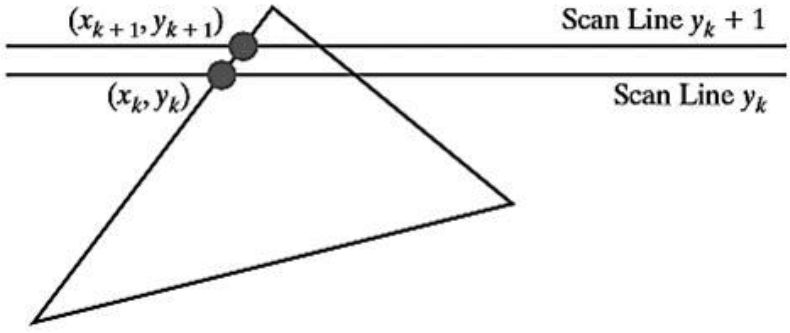
7.1 Scanline Polygon Filling

Scanline filling is basically filling up of polygons using horizontal lines or scanlines. The purpose of this algorithm is to fill (color) the interior pixels of a polygon given only the vertices of the figure. To understand Scanline, think of the image being drawn by a single pen starting from bottom left, continuing to the right, plotting only points where there is a point present in the image, and when the line is complete, start from the next line and continue. This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections. With reference to the figure given bewlo, following special cases of polygon vertices are considered if the scanline passes though vertex:

1. If both lines intersecting at the vertex are on the same side (either above or below) of the scanline, consider it as two points. (Here, lines passing through B, C, D and F)
2. If lines intersecting at the vertex are at opposite sides of the scanline, consider it as only one point. (Here, lines passing through A, E)



Here, coherence property is used for calculating next pixel on the scan line. Following figure explains the same.



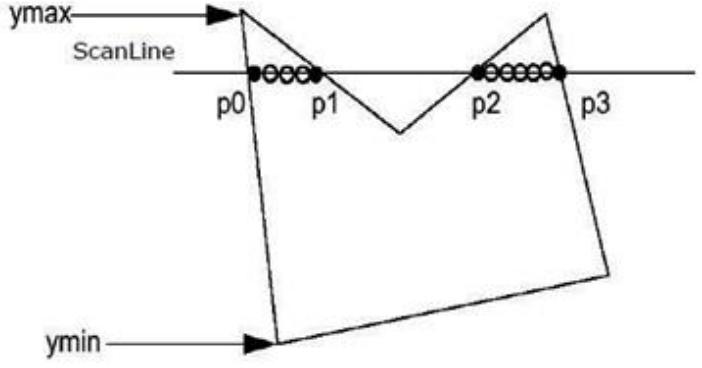
Slope of the edge can be given by

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

Change in y coordinates between the two scan lines will be 1. So given the current x-intercept, the next x-intercept coordinate $x_{k+1} = x_k + (1/m)$.

Algorithm

1. Find out the Y_{min} and Y_{max} from the given polygon.

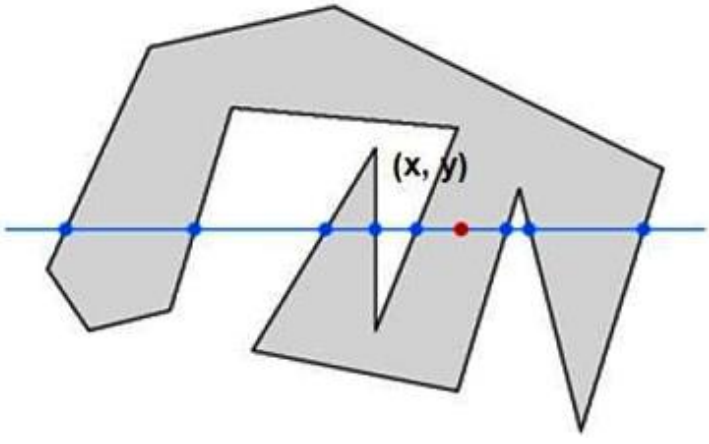


2. ScanLine intersects with each edge of the polygon from Y_{min} to Y_{max}. Name each intersection point of the polygon. As per the figure shown above, they are named as p_0, p_1, p_2, p_3.
3. Sort the intersection point in the increasing order of X coordinate i.e. p_0p_1, p_1p_2 and p_2p_3.
4. Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

7.2 Inside-Outside Test for Scanlines

While filling an object, we often need to identify whether particular point is inside the object or outside it. This method is also known as counting number method. There are two methods by which we can identify whether particular point is inside an object or outside.

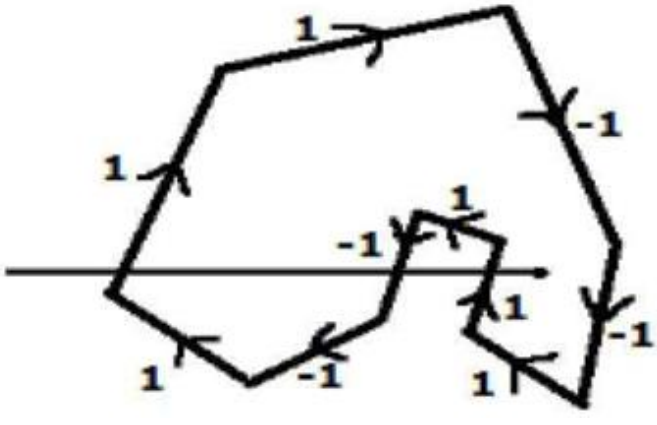
1. Odd-Even Rule



In this technique, we count the edge crossing along the line from any point (x, y) to infinity, on both the sides. If the number of intersections is odd on each side of the point, then the point (x,y) is an interior point; and if the number of interactions is even, then the point (x,y) is an exterior point. In the above figure, we can see that from the point (x, y) the number of intersections point on the left side is 5 and on the right side is 3. On both the sides, the number of intersection points are odd, so the point is considered within the object. The same special cases are considered if the scanline passes though vertex:

1. If both lines intersecting at the vertex are on the same side (either above or below) of the scanline, consider it as two points.
2. If lines intersecting at the vertex are at opposite sides of the scanline, consider it as only one point.

2. Nonzero Winding Number Rule



This method is also used with the simple polygons to test the given point is interior or not. In this method, first of all, give directions to all the edges of the polygon. Draw a scan line from the point to be tested towards the left most of X direction.

1. Give the value 1 to all the edges which are going to upward direction and all other -1 as direction values.
2. Check the edge direction values from which the scan line is passing and sum up them.
3. If the total sum of this direction value is non-zero, then this point to be tested is an interior point, otherwise it is an exterior point.

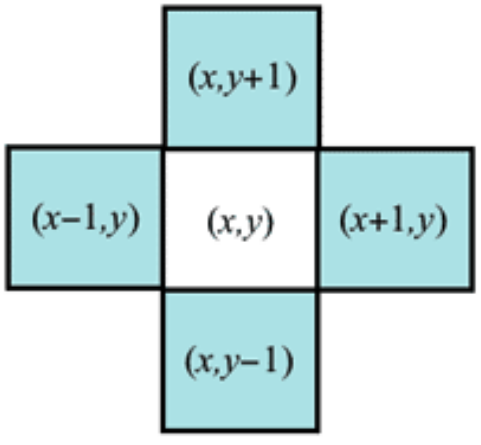
In the above figure, we sum up the direction values from the line through which the scan line is passing then the total is 1 – 1 + 1 = 1 which is non-zero. So the point is said to be an interior point.

7.3 Boundary Fill Algorithm

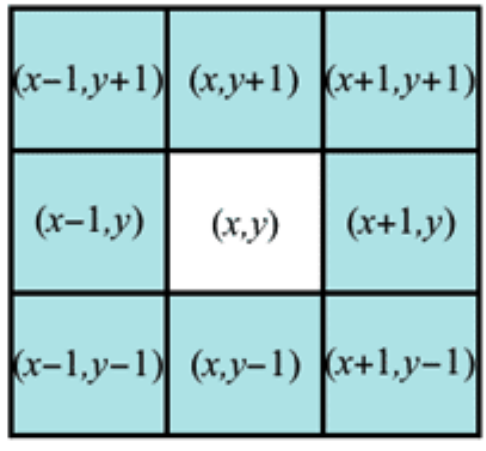
The boundary fill algorithm works as per its name. This algorithm picks a point inside an object and starts to fill until it hits the boundary of the object. The color of the boundary and the color that we fill-in should be different for this algorithm to work. In this algorithm, we assume that color of the boundary is same for the entire object. The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

4-Connected and 8-Connected Pixels

In this technique 4-connected pixels are used as shown in the figure. We are putting the pixels above, below, to the right, and to the left side of the current pixels and this process continues until we find a boundary with different color. In 8-connected pixels, addition to 4-connected pixels, we are also putting pixels in diagonals so that entire area of the current pixel is covered. This process again continues until we find a boundary with different color. Both approaches are shown in figure below:



4-Connected Pixel



8-Connected Pixels

Algorithm (4-Connected)

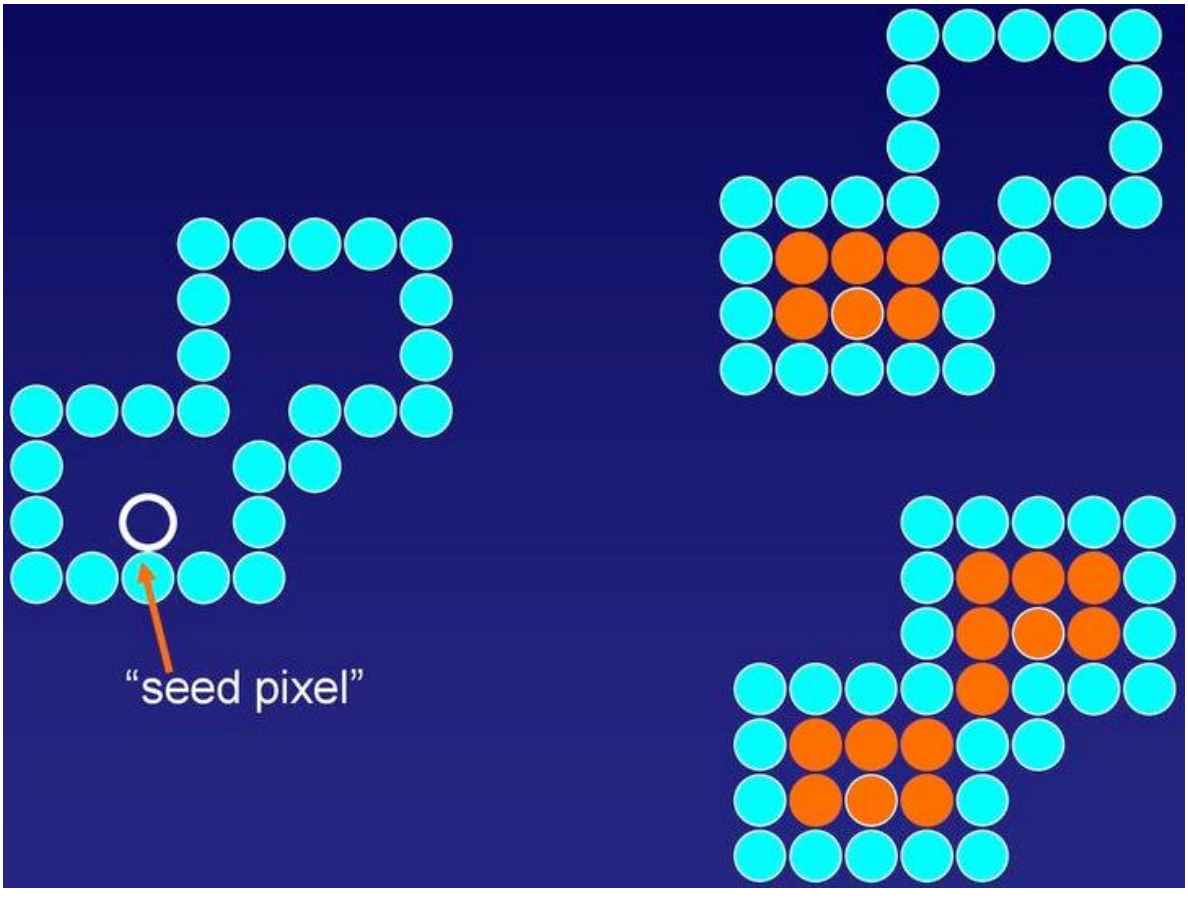
```
void boundaryFill4(int x, int y, int fill_color,int boundary_color)
{
   if(getpixel(x, y) != boundary_color && getpixel(x, y) != fill_color)
   {
            putpixel(x, y, fill_color);
            boundaryFill4(x + 1, y, fill_color, boundary_color);
            boundaryFill4(x - 1, y, fill_color, boundary_color);
            boundaryFill4(x, y + 1, fill_color, boundary_color);
            boundaryFill4(x, y - 1, fill_color, boundary_color);
   }
}
```

Algorithm (8-Connected)

```
void boundaryFill8(int x, int y, int fill_color,int boundary_color)
{
   if(getpixel(x, y) != boundary_color && getpixel(x, y) != fill_color)
   {
      putpixel(x, y, fill_color);
      boundaryFill8(x + 1, y, fill_color, boundary_color);
      boundaryFill8(x - 1, y, fill_color, boundary_color);
      boundaryFill8(x, y + 1, fill_color, boundary_color);
      boundaryFill8(x, y - 1, fill_color, boundary_color);
      boundaryFill8(x - 1, y - 1, fill_color, boundary_color);
      boundaryFill8(x + 1, y + 1, fill_color, boundary_color);
      boundaryFill8(x - 1, y + 1, fill_color, boundary_color);
      boundaryFill8(x + 1, y - 1, fill_color, boundary_color);
   }
}
```

4-connected pixels Vs 8-connected pixels

Let us consider the following figure with the boundary color as LIGHT and the fill color as DARK. The 4-connected method fails to fill this figure completely as shown on top right side. This figure will be efficiently filled using the 8-connected technique as it is shown on bottom right.



7.4 Flood Fill Algorithm

Sometimes we come across an object where we want to fill the area and its boundary with different colors. We can paint such objects with a specified interior color instead of searching for particular boundary color as in boundary filling algorithm. Instead of relying on the boundary of the object, it relies on the fill color. In other words, it replaces the interior color of the object with the fill color. When no more pixels of the original interior color exist, the algorithm is completed. This algorithm relies on the Four-connect or Eight-connect method of filling in the pixels. It looks for 4 or 8 adjacent pixels that are a part of the interior as under:

Algorithm (4-Connected)

1. Initialize the value of seed point seedx, seedy, fcolor and dcol.
2. Define the boundary values of the polygon.
3. Check if the current seed point is of default color, then repeat the steps 4 and 5 till the boundary pixels reached.

   If getpixel(x, y) = dcol then repeat step 4 and 5
4. Change the default color with the fill color at the seed point.

   setPixel(seedx, seedy, fcol)
5. Recursively follow the procedure with four neighborhood points.

   FloodFill (seedx – 1, seedy, fcol, dcol)

   FloodFill (seedx + 1, seedy, fcol, dcol)

   FloodFill (seedx, seedy - 1, fcol, dcol)

   FloodFill (seedx, seedy + 1, fcol, dcol)

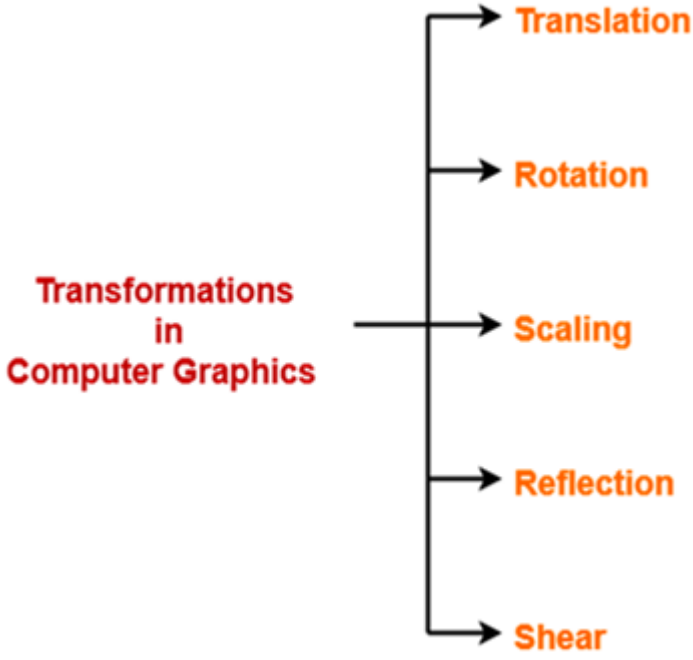
EXERCISE

1. Calculate the points between the starting point (5, 6) and ending point (8, 12). Apply the Digital Differential Analyzer algorithm to plot a line.
2. Calculate the points between the starting coordinates (20, 10) and ending coordinates (30, 18) using Bresanham's Algorithm.
3. Calculate the points between the starting coordinates (5, 9) and ending coordinates (12, 16) using Miidpoint line algorithm.
4. Derive Midpoint Circle algorithm from starting point (r,0).
5. Derive Midpoint Ellipse algorithm with starting point (0, r_y) for Region R1 and starting point (r_x,0) for Region R2.
6. Derive Midpoint Ellipse algorithm with starting point (r_x,0) and ending point (x,y) of region R1. The starting point of region R2 will be ending point (x,y) of R1 and ending point of region R2 will be (0, r_y).
7. Differentiate the effect of 4-connected and 8-connected pixels in flood fill and boundary fill algorithms.
8. Derive 8-Connected Flood Fill Algorithm.

3. 2D TRANSFORMATIONS

1. WHAT IS TRANSFORMATION?

In computer graphics sometimes objects need to be manipulated. That is, we may want to reposition them or rotate them or resize them. We can achieve this with the help of basic transformations. Transformation refers to the mathematical operations or rules that are applied on a graphical image consisting of the number of lines, circles, and ellipses to change its size, shape, or orientation. It can also reposition the image on the screen. Transformations play a very crucial role in computer graphics. There are various types of transformations in computer graphics through which an image can be processed, edited ad altered. Some basic and most commonly used types of these transformations are:



2. TRANSLATION

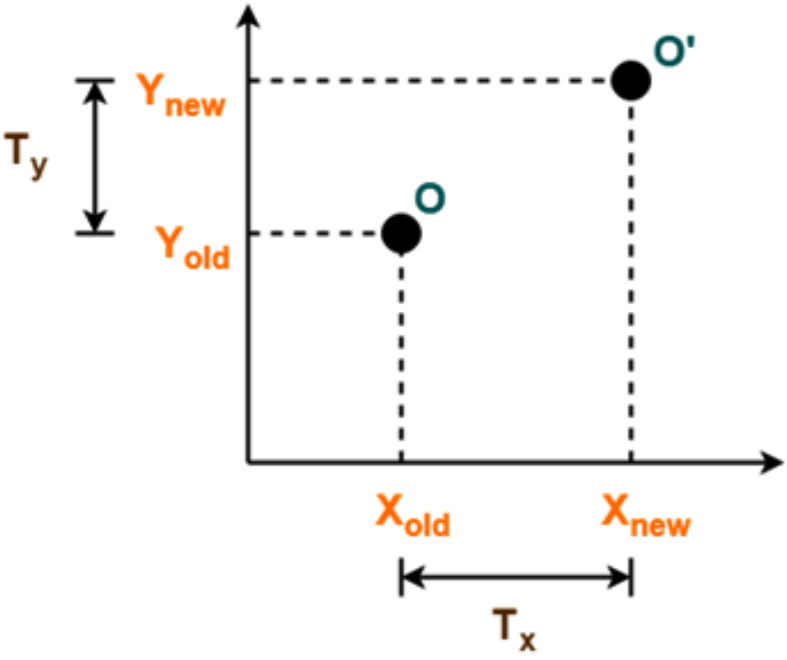
Repositioning along a straight line path from one location to another are called 2D translation. That is, we translate a two-dimensional point by adding translation distances, T_x and T_y to the original coordinate position (x, y) to move the point to a new position (x', y'). Consider a point object O has to be moved from one position to another in a 2D plane.

Let-

- Initial coordinates of the object O = (X_{old}, Y_{old})
- New coordinates of the object O after translation = (X_{new}, Y_{new})
- Translation vector or Shift vector = (T_x, T_y)

Given a Translation vector (T_x, T_y)

- T_x defines the distance the X_{old} coordinate has to be moved.
- T_y defines the distance the Y_{old} coordinate has to be moved.



This translation is achieved by adding the translation coordinates to the old coordinates of the object as

- $X_{new} = X_{old} + T_x$ (This denotes translation towards X axis)
- $Y_{new} = Y_{old} + T_y$ (This denotes translation towards Y axis)

- If T_x is positive, object will move in right direction
- If T_x is negative, object will move towards left
- If T_y is positive object will move upwards
- if T_y is negative , object will move downwards.

In Matrix form, the above translation equations may be represented as

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

Translation Matrix

- The homogeneous coordinates representation of (X, Y) is (X, Y, 1).
- Through this representation, all the transformations can be performed using matrix / vector multiplications.

We will discuss about homogenous representation in section 7.1. The above translation matrix may be represented as a 3 x 3 matrix as in homogeneous coordinates

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

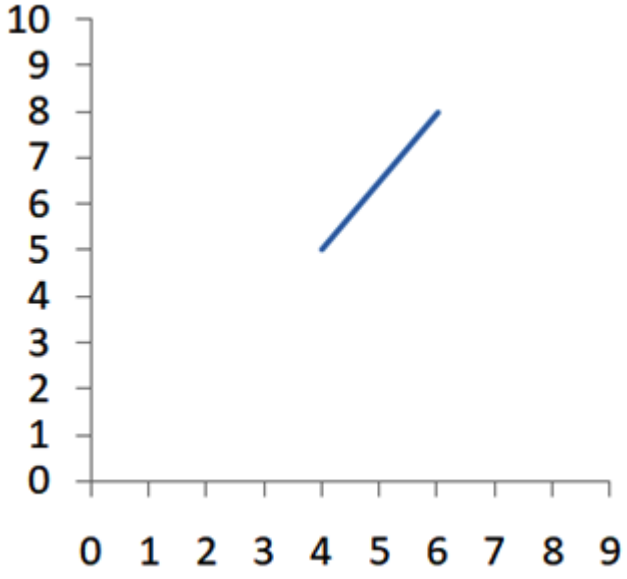
Translation Matrix using homogenous coordinates

Suppose there is a point (3, 4) and we add translation vector (2, 3). Then the new point will be (3+2, 4+3) i.e.(5, 7). 2D Translation is a rigid-body transformation that moves objects without deformation. To translate a Line we translate line endpoints one by one. However, Polygons are translated by adding the translation vector to the coordinate position of each vertex and we redraw the polygon.

Example 1: Translate a line AB A(4,5), B(6,8) 2 units in x direction and 3 units in y direction.

Solution:

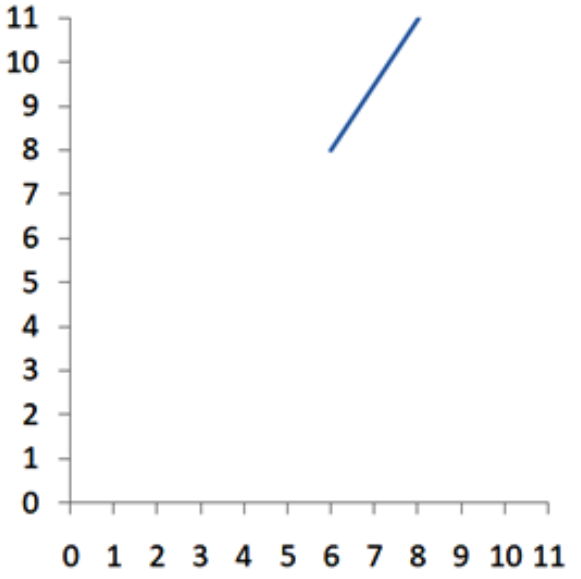
Given : x1 = 4 , y1 = 5 , x2 = 6 , y2 = 8, tx = 2 , ty = 3



Original line

x1′ = x1+tx = 4+2 = 6
y1′ = y1+ty = 5+3 = 8
x2′ = x2+tx = 6+2 = 8
y2′ = y2+ty = 8+3 = 11

New coordinates after translation, A'B' A'(6, 8) , B'(8, 11)



Line after translation

Example 2: Given a circle C with radius 10 and center coordinates (1, 4). Apply the translation with distance 5 towards X axis and 1 towards Y axis. Obtain the new coordinates of C without changing its radius.

Solution:

Given that

- Old center coordinates of C = $(X_{old}, Y_{old}) = (1, 4)$
- Translation vector = $(T_x, T_y) = (5, 1)$

Let the new center coordinates of C = (X_{new}, Y_{new}).

Applying the translation equations, we have-

- $X_{new} = X_{old} + T_x = 1 + 5 = 6$
- $Y_{new} = Y_{old} + T_y = 4 + 1 = 5$

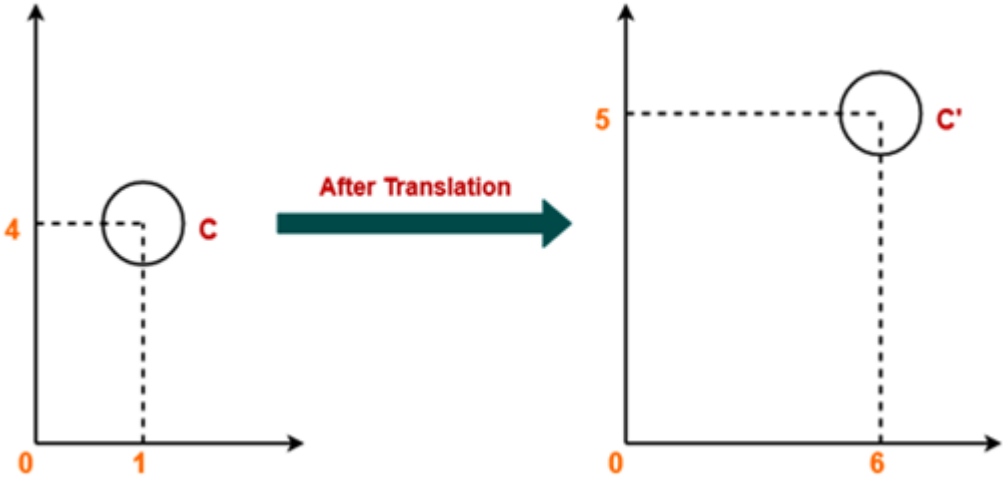
Thus, New center coordinates of C = (6, 5).

Using matrix, the operation can be performed as under:

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix} + \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \end{bmatrix}$$

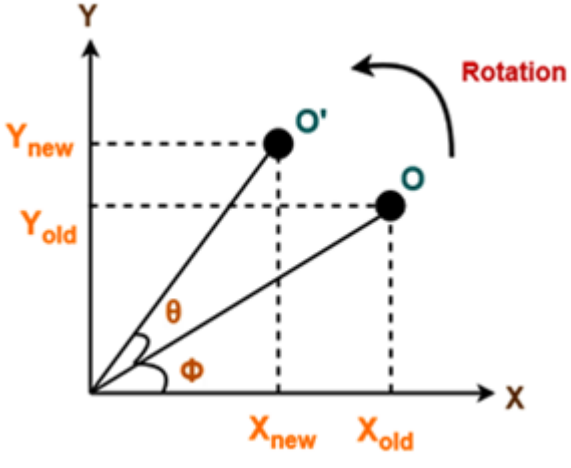


3. ROTATION

Consider a point object O has to be rotated from one angle to another in a 2D plane. Let, the initial coordinates of the object O = (X_{old}, Y_{old}), initial angle of the object O with respect to origin = Φ, Rotation angle = θ and the new coordinates of the object O after rotation = (X_{new}, Y_{new})

Two types of Rotations are Possible:

1. Anticlockwise Rotation
2. Clockwise Rotation



Using standard trigonometric the original coordinate of point O(X,Y) can be represented as,

$$x = r\cos\phi \quad .....(1)$$
$$y = r\sin\phi \quad .....(2)$$

Same way we can represent the point O'(X′,Y′) as,

$$x' = r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta \quad .....(3)$$
$$y' = r\sin(\phi+\theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta \quad .....(4)$$

Substituting equation 1 & 2 in 3 & 4 respectively, we will get

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

Representing the above equation in matrix form,

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

Using homogenous coordinates, it can be represented as

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

For, anti-clockwise Rotation, the angle is positive while for clockwise Rotation angle is negative. So, for clockwise rotations, θ can be replaced by -θ and the matrix becomes,

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

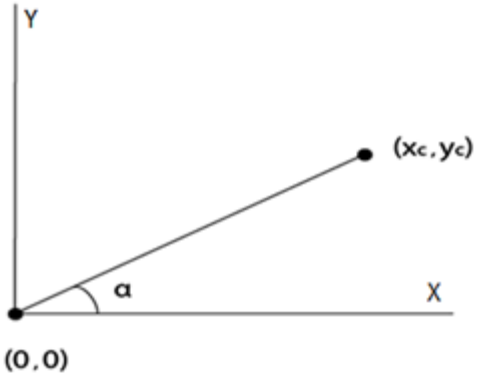
When the object is rotated, then every point of the object is rotated by the same angle. Straight Line is rotated by the endpoints with the same angle and redrawing the line between new endpoints. Polygon is rotated by shifting every vertex using the same rotational angle. Ellipse rotation can be obtained by rotating major and minor axis of an ellipse by the desired angle.

3.1 Rotation about an arbitrary point

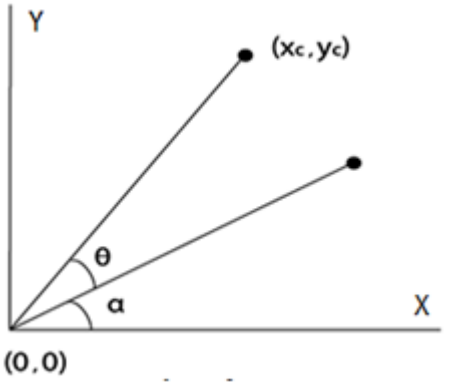
If we want to rotate an object or point about an arbitrary point, first of all, translate the point about which we want to rotate to the origin. Then rotate point or object about the origin,

and at the end, again translate it to the original place. This way we will get rotation about an arbitrary point. For example, the point (x, y) is to be rotated and (x_c y_c) is an arbitrary point about which anti-clockwise rotation is done

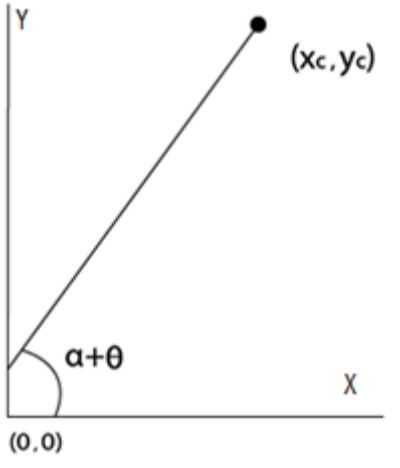
Step1: Translate point (x_c y_c) to origin



Step2: Rotation of (x, y) about the origin

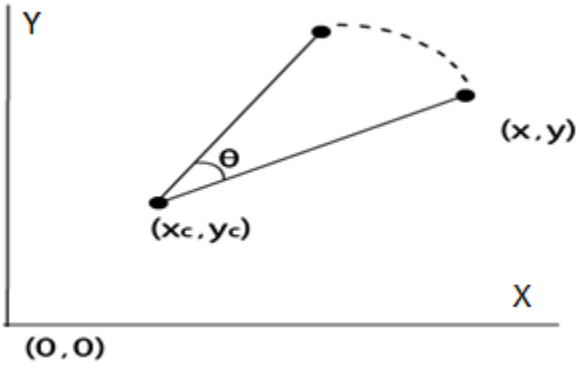


Step3: Translation of center of rotation back to its original position



Step4: Finally, the translation of center of rotation back to its original position

---



Example 1: Given a line segment with starting point as (0, 0) and ending point as (4, 4). Apply 30 degree rotation anticlockwise direction on the line segment and find out the new coordinates of the line.

Solution:

We rotate a straight line by its end points with the same angle. Then, we redraw a line between the new end points.

Given-

- Old ending coordinates of the line = $(X_{old}, Y_{old}) = (4, 4)$
- Rotation angle = $\theta = 30^o$

Let new ending coordinates of the line after rotation = (X_{new}, Y_{new}).

Applying the rotation equations, we have-

$X_{new} = X_{old} \times \cos\theta - Y_{old} \times \sin\theta$

$= 4 \times \cos30^o - 4 \times \sin30^o$

$= 4 \times (\sqrt{3} / 2) - 4 \times (1 / 2)$

$= 2\sqrt{3} - 2$

$= 2(\sqrt{3} - 1)$

$= 2(1.73 - 1)$

$= 1.46$

$Y_{new} = X_{old} \times \sin\theta + Y_{old} \times \cos\theta$

$= 4 \times \sin30^o + 4 \times \cos30^o$

$= 4 \times (1 / 2) + 4 \times (\sqrt{3} / 2)$

$= 2 + 2\sqrt{3}$

$= 2(1 + \sqrt{3})$

$= 2(1 + 1.73)$

$= 5.46$

Thus, New ending coordinates of the line after rotation = (1.46, 5.46).

Alternatively,

In matrix form, the new coordinates of the line after rotation may be obtained as

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

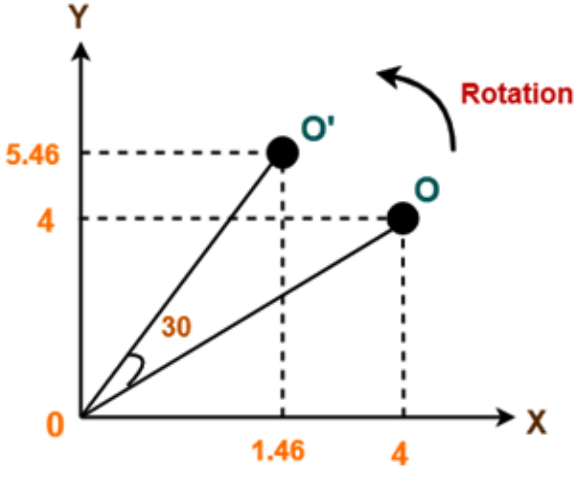
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos 30 & -\sin 30 \\ \sin 30 & \cos 30 \end{bmatrix} \times \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 4 \times \cos 30 - 4 \times \sin 30 \\ 4 \times \sin 30 + 4 \times \cos 30 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 4 \times \cos 30 - 4 \times \sin 30 \\ 4 \times \sin 30 + 4 \times \cos 30 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1.46 \\ 5.46 \end{bmatrix}$$

Thus, New ending coordinates of the line after rotation = (1.46, 5.46).



Example 2: Given a triangle with corner coordinates (0, 0), (1, 0) and (1, 1). Rotate the triangle by 90 degree anticlockwise direction and find out the new coordinates.

Solution:

We rotate a polygon by rotating each vertex of it with the same rotation angle.
Given that

- Old corner coordinates of the triangle = A (0, 0), B(1, 0), C(1, 1)
- Rotation angle = $\theta = 90^o$

For Coordinates A(0, 0)

Let the new coordinates of corner A after rotation = (X_{new}, Y_{new}).
Applying the rotation equations, we have-

$X_{new} = X_{old} \times \cos\theta - Y_{old} \times \sin\theta$
$= 0 \times \cos 90^o - 0 \times \sin 90^o$
$= 0$

$Y_{new} = X_{old} \times \sin\theta + Y_{old} \times \cos\theta$
$= 0 \times \sin 90^o + 0 \times \cos 90^o$
$= 0$

Thus, New coordinates of corner A after rotation = (0, 0).

For Coordinates B(1, 0)

Let the new coordinates of corner B after rotation = (X_{new}, Y_{new}).

$X_{new} = X_{old} \times \cos\theta - Y_{old} \times \sin\theta$
$= 1 \times \cos 90^o - 0 \times \sin 90^o$
$= 0$

$Y_{new} = X_{old} \times \sin\theta + Y_{old} \times \cos\theta$
$= 1 \times \sin 90^o + 0 \times \cos 90^o$
$= 1 + 0$
$= 1$

Thus, New coordinates of corner B after rotation = (0, 1).

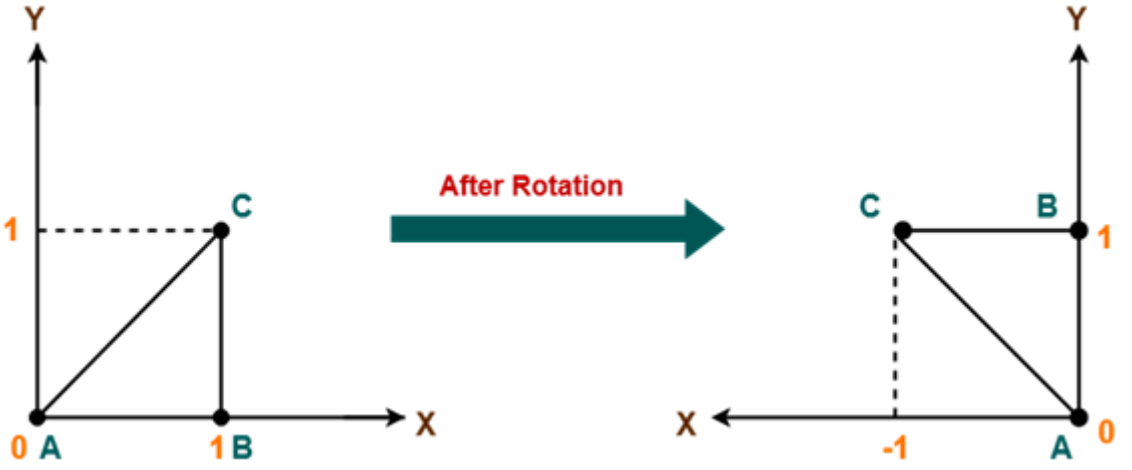
For Coordinates C(1, 1)

Let the new coordinates of corner C after rotation = (X_{new}, Y_{new}).

$X_{new} = X_{old} \times \cos\theta - Y_{old} \times \sin\theta$
$= 1 \times \cos 90^o - 1 \times \sin 90^o$
$= 0 - 1$
$= -1$

$Y_{new} = X_{old} \times \sin\theta + Y_{old} \times \cos\theta$
$= 1 \times \sin 90^o + 1 \times \cos 90^o$
$= 1 + 0$
$= 1$

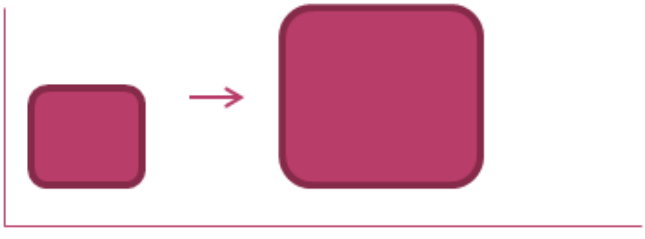
Thus, New coordinates of corner C after rotation = (-1, 1).
Thus, New coordinates of the triangle after rotation = A (0, 0), B(0, 1), C(-1, 1).



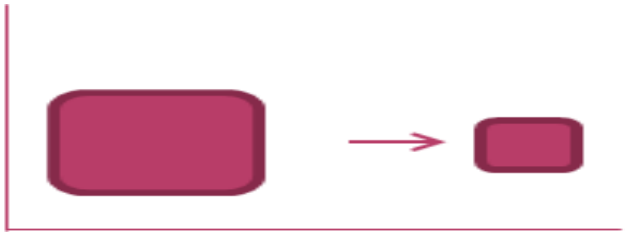
4. SCALING

2D Scaling Transformation means changing the size of an object. It is a basic geometric transformation. We can scale the object by multiplying scaling factors with coordinate points. This Scaling Transformation is about the origin.

- Scaling may be used to increase or reduce the size of object.
- Scaling subjects the coordinate points of the original object to change.
- Scaling factor determines whether the object size is to be increased or reduced.
- If scaling factor > 1, then the object size is increased.



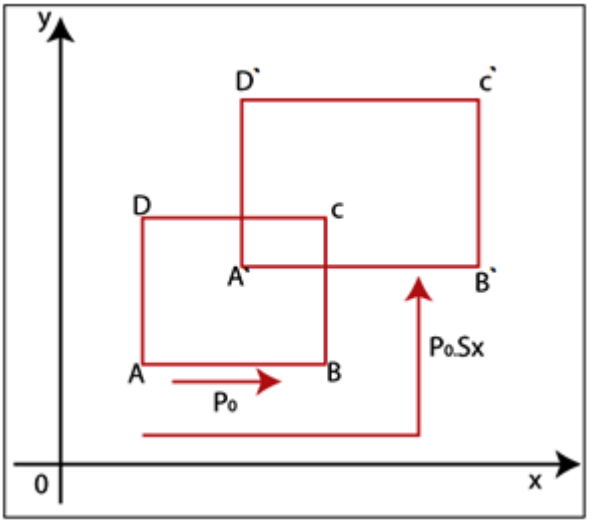
- If scaling factor < 1, then the object size is reduced.



For Example, as given in following figure, If we want to scale an object that has R (P0, Q0) coordinate and the new coordinates of an object are R` (P1, Q1) then the equation will be-

$P1 = P_0 . S_x$

$Q1 = Q_0 . S_y$



Consider a point object O has to be scaled in a 2D plane. Let,

- Initial coordinates of the object O = (X_{old}, Y_{old})
- Scaling factor for X-axis = S_x
- Scaling factor for Y-axis = S_y
- New coordinates of the object O after scaling = (X_{new}, Y_{new})

This scaling is achieved by using the following scaling equations-

- $X_{new} = X_{old} \times S_x$
- $Y_{new} = Y_{old} \times S_y$

Representing it in matrix form,

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

There are 2 types of scaling:

1. **Uniform Scaling:** When both the scaling factors S_x, S_y are same. i.e. S_x=S_y. In this case Size increases or reduces uniformly in both the directions. That is ratio of scaling of the object remains same in both the dimensions. for example in the following figure square remains square.
2. **Differential Scaling:** When both the scaling factors S_x, S_y are different. Scaling will not be same in both x and y direction. In the following figure increase in x direction is more as compared to y direction. Therefore square has transformed to rectangle.

For homogeneous coordinates, the above scaling matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} x \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Example: Given a square object with coordinate points A(0, 3), B(3, 3), C(3, 0), D(0, 0). Apply the scaling parameter 2 towards X axis and 3 towards Y axis and obtain the new coordinates of the object.

Solution:

- Old corner coordinates of the square = A (0, 3), B(3, 3), C(3, 0), D(0, 0)
- Scaling factor along X axis = 2
- Scaling factor along Y axis = 3

For Coordinates A(0, 3)

Let the new coordinates of corner A after scaling = (X_{new}, Y_{new}).
Applying the scaling equations,

- $X_{new} = X_{old} \times S_x = 0 \times 2 = 0$
- $Y_{new} = Y_{old} \times S_y = 3 \times 3 = 9$

Thus, New coordinates of corner A after scaling = (0, 9).
For Coordinates B(3, 3)
Let the new coordinates of corner B after scaling = (X_{new}, Y_{new}).
Applying the scaling equations,

- $X_{new} = X_{old} \times S_x = 3 \times 2 = 6$
- $Y_{new} = Y_{old} \times S_y = 3 \times 3 = 9$

Thus, New coordinates of corner B after scaling = (6, 9).

For Coordinates C(3, 0)

Let the new coordinates of corner C after scaling = (X_{new}, Y_{new}).
Applying the scaling equations,

- $X_{new} = X_{old} \times S_x = 3 \times 2 = 6$
- $Y_{new} = Y_{old} \times S_y = 0 \times 3 = 0$

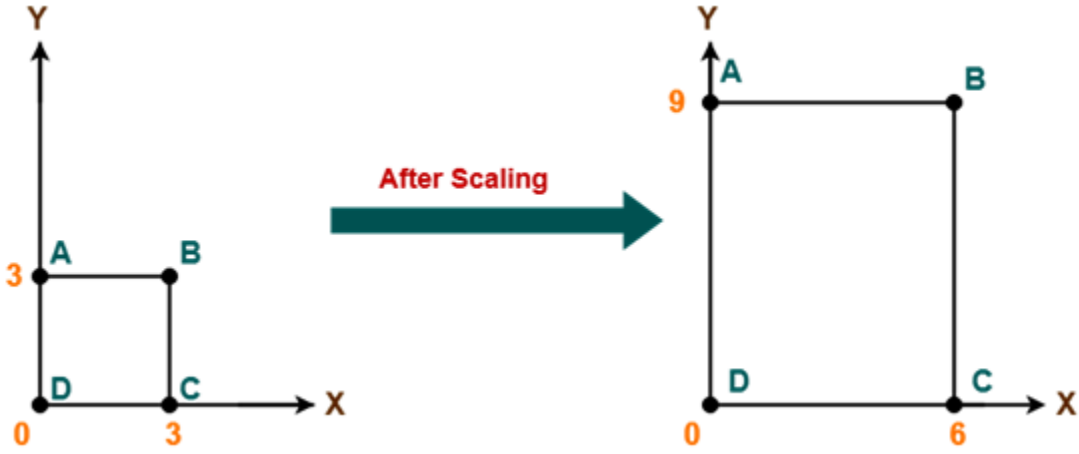
Thus, New coordinates of corner C after scaling = (6, 0).

For Coordinates D(0, 0)

Let the new coordinates of corner D after scaling = (X_{new}, Y_{new}).
Applying the scaling equations, we have-

- $X_{new} = X_{old} \times S_x = 0 \times 2 = 0$
- $Y_{new} = Y_{old} \times S_y = 0 \times 3 = 0$

Thus, New coordinates of corner D after scaling = (0, 0).



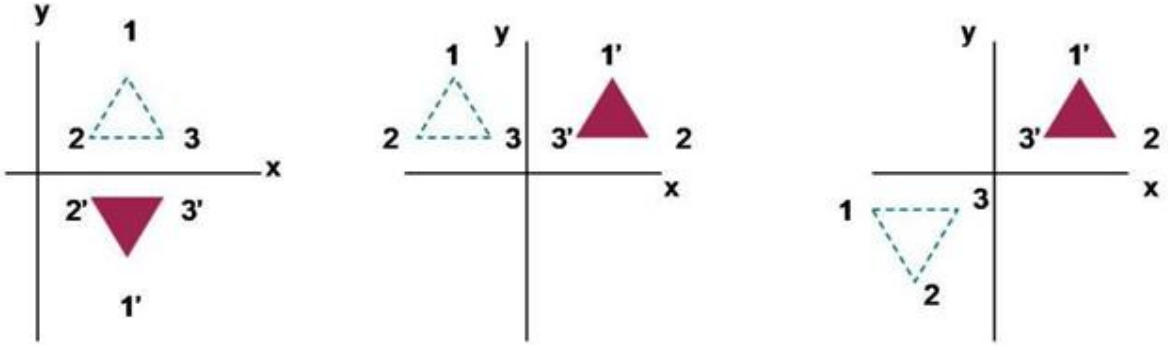
5. REFLECTION

Reflection is a transformation that produces a mirror image of an object. 2D Reflection is performed about a line. This line can be one of the principal axis or it can be any arbitrary line. Reflection is actually rotating an object 180 degrees about the reflection axis. Following are the types of reflections.

- Reflection About the line y=0 i.e. x-axis
- Reflection About the line x=0 i.e. y-axis
- Reflection About the axis perpendicular to XY plane and which passes through the coordinate origin
- Reflection About the line y=x
- Reflection About any arbitrary line y=mx+b

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

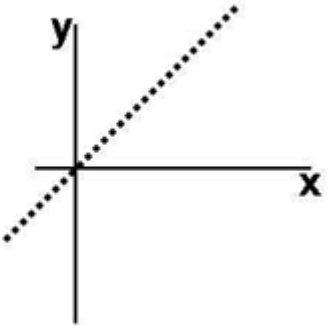
x-axis y-axis origin or xy plane



As you can see, When 2D Reflection is performed about the x-axis. The object will be reflected from the First quadrant to the Fourth quadrant or vice-versa. If the original object is in the second quadrant final object will be in the third quadrant. It works as if the x-axis is a mirror.

5.1 Reflection with respect to a Line y=x

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



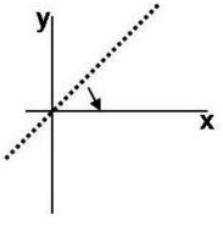
We know how to perform reflection about the x-axis. So, we can derive 2D Reflection about a line y=x by a sequence of operations.

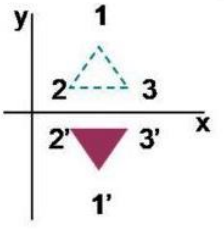
- First, perform Clockwise rotation by 45 degrees. So that this line y=x coincides with the x-axis.
- After that, perform Reflection about the x-axis.
- Now, send the object back to its original position. Therefore, we will perform Counter clockwise rotation by 45 degrees.

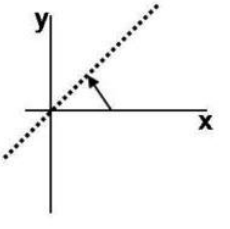
This sequence of operations can be performed with the help of composite transformations.

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

$$\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



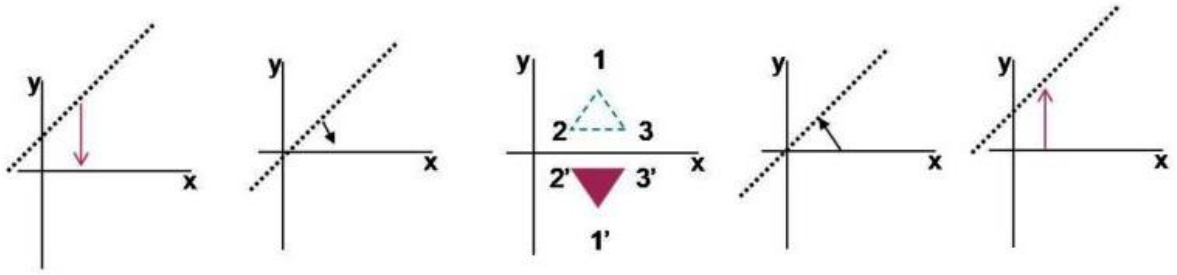




5.2 Reflection with respect to an arbitrary Line y=mx+b

We can reflect an object about any given line. We can easily derive the Reflection matrix by performing following series of operations.

- Translate the line so that it passes through the origin: It is required that the line should pass through the origin. As all the axis pass through the origin. We will translate the object by the factor (0, -b), where b is the y-intercept of the line. So, it is T(0,-b)
- Clockwise rotation: Now we have to change the orientation so that line passes through any of the axis. We can find the angle of rotation by the relationship m = tanΘ where m is the slope of the line. So, it is R(-Θ)
- Reflection about that axis: Here, we will simply reflect the object about the axis using a standard 2D Reflection matrix.
- Counter clockwise rotation by the same angle: We will perform R(Θ) to change the orientation back as it was original.
- Translate back to original position: We will translate the object with the same T(0,-b) factor but in opposite direction i.e. T(0,b).



Example 1: There is a triangle ABC A(-1,-1) B(0,-2)C(1,-1). Reflect the image about x axis.

Solution:

The object matrix

$$O = \begin{bmatrix} x1 & x2 & x3 \\ y1 & y2 & y3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & -2 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

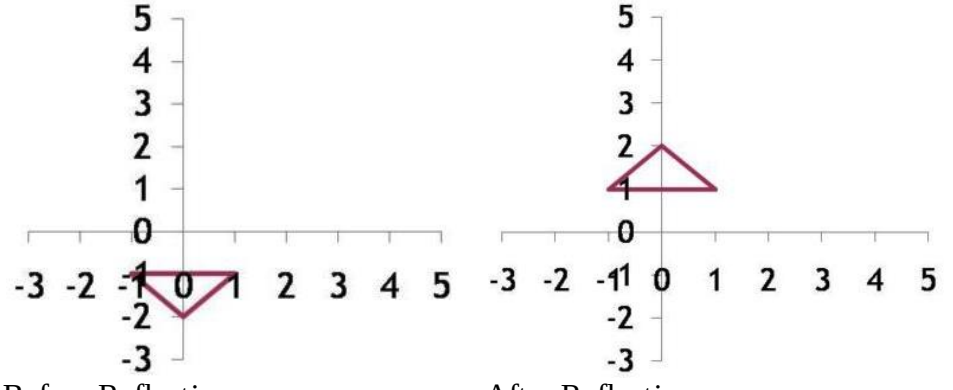
The Reflection matrix

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

Finally, the solution matrix is,

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

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



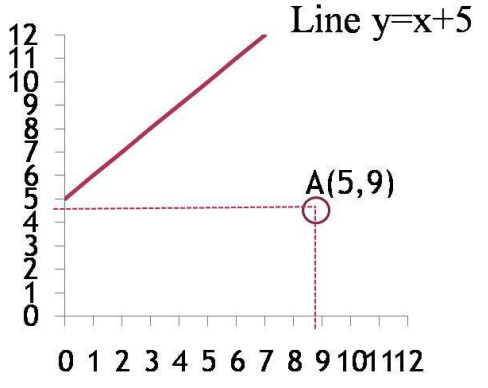
Before Reflection After Reflection

Example 2: Find the reflection of a point A(5,9) about the line y=x+5.

Solution:

We have to follow the sequence of operations.

- Translate the line so that it passes through the origin by T(0,-5)
- This is the diagonal line y=x. Reflect on the line by y=x
- Translate the line back to original position by T(0,5)



Object Matrix

$$O = \begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 9 \\ 1 \end{bmatrix}$$

Translation Matrix

$$T(0, -5) = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -5 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflection Matrix

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

Translation Matrix

$$T(0, 5) = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix}$$

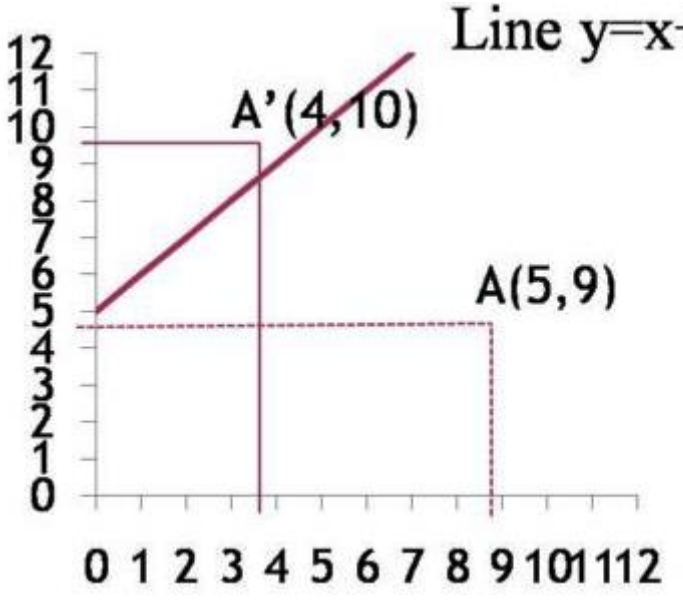
Finally, arrange them as O' = T(0, 5). Ref. T(0, -5). O and solve.

$$O' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 9 \\ 1 \end{bmatrix}$$

$$O' = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 9 \\ 1 \end{bmatrix}$$

$$O' = \begin{bmatrix} 0 & 1 & -5 \\ 1 & 0 & 5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 9 \\ 1 \end{bmatrix} \qquad O' = \begin{bmatrix} 4 \\ 10 \\ 1 \end{bmatrix}$$

So, final answer is A'(4,10). If we plot original and final points on the graph. We get,



6. SHEAR

A transformation that distorts the shape of an object is Shear Transformation. It appears like somebody has dragged the object in a specific direction and its layers appear to slide. For Shear Transformation we need three parameters.

- Direction of Shear
- The magnitude of Shear or Force of Shear.
- Axis of Shear

The direction of Shear is the direction in which we apply Shear Transformation. We denote the magnitude of shear as Sh_x or Sh_y. It is the amount of shear that we apply to the object. The Axis of shear is the axis with respect to which we perform Shear. We can think of axis of shear as the firm base to which the object is attached.

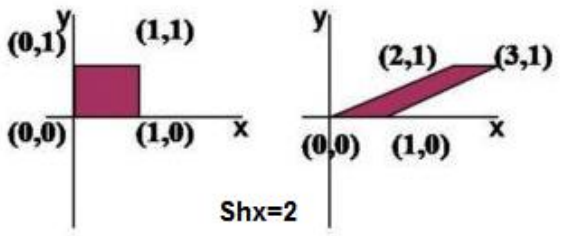
6.1 X-Axis Shear

X-axis Shear is given by:

$x' = x + y\ sh_x$

$y' = y$

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



- As a result of Shear, Square is converted to parallelogram.
- There is a shift in the x-direction by a factor of $y.sh_x$. While there is no change in y coordinate.
- As we can see from the equation the shear effect also depends on the distance of the y coordinate from the origin ($y.sh_x$). The greater the distance, the greater is the effect of shear.

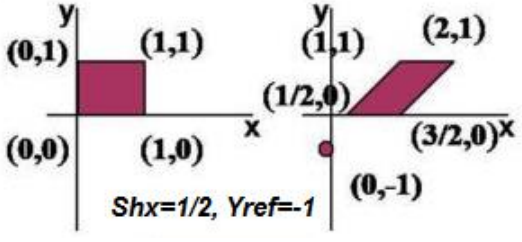
6.2 Shear with respect to line y_{ref}

We can also perform shear with respect to a line parallel to x-axis.

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

$y' = y$

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



- Square is converted to a Shifted parallelogram.
- There is a shear effect and as well as translation (shifting of the object).
- There is a translation by the amount $-sh_x.y_{ref}$. In the above example, y_{ref} is -1. So overall translation factor becomes positive. And we get a shift towards the right. If it is positive. Then translation factor will be negative and we will get a shift towards the left.

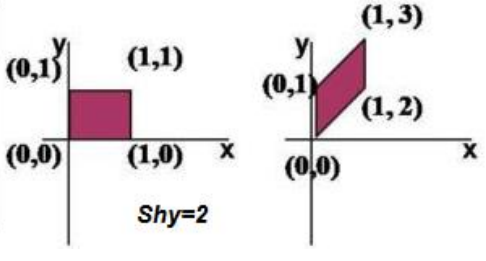
6.3 Y-Axis Shear

We can perform Shear Transformation with respect to y-axis. x coordinate value remains unaffected.

$x' = x$

$y' = y + x\, sh_y$

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



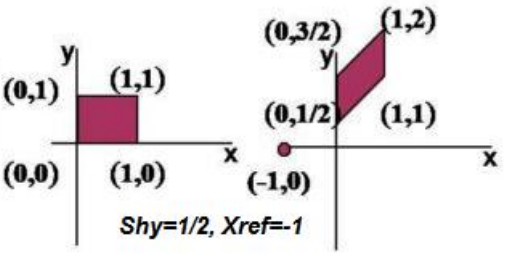
6.4 Shear with respect to line x_{ref}

We can perform Shear with respect to any line parallel to the y-axis. As a result, there will be a slight shift along with the shear effect. Also, the shift will be upwards or downwards will depend on the sign of x_{ref} (positive or negative).

$x' = x$

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

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



Example: Given a triangle with points (1, 1), (0, 0) and (1, 0). Apply shear parameter 2 on X axis and 2 on Y axis and find out the new coordinates of the object.

Solution:

Given that,

- Old corner coordinates of the triangle = A (1, 1), B(0, 0), C(1, 0)
- Shearing parameter towards X direction (Sh_x) = 2
- Shearing parameter towards Y direction (Sh_y) = 2

Shearing in X Axis:

For Coordinates A(1, 1)

Let the new coordinates of corner A after shearing = (X_{new}, Y_{new}).
Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Y_{old} = 1 + 2 \times 1 = 3$
- $Y_{new} = Y_{old} = 1$

Thus, New coordinates of corner A after shearing = (3, 1).

For Coordinates B(0, 0)

Let the new coordinates of corner B after shearing = (X_{new}, Y_{new}).
Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Y_{old} = 0 + 2 \times 0 = 0$
- $Y_{new} = Y_{old} = 0$

Thus, New coordinates of corner B after shearing = (0, 0).

For Coordinates C(1, 0)

Let the new coordinates of corner C after shearing = (X_{new}, Y_{new}).
Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Y_{old} = 1 + 2 \times 0 = 1$
- $Y_{new} = Y_{old} = 0$

Thus, New coordinates of corner C after shearing = (1, 0).
Thus, New coordinates of the triangle after shearing in X axis = A (3, 1), B(0, 0), C(1, 0).
Shearing in Y Axis:

For Coordinates A(1, 1)

Let the new coordinates of corner A after shearing = (X_{new}, Y_{new}).
Applying the shearing equations, we have-

- $X_{new} = X_{old} = 1$
- $Y_{new} = Y_{old} + Sh_y \times X_{old} = 1 + 2 \times 1 = 3$

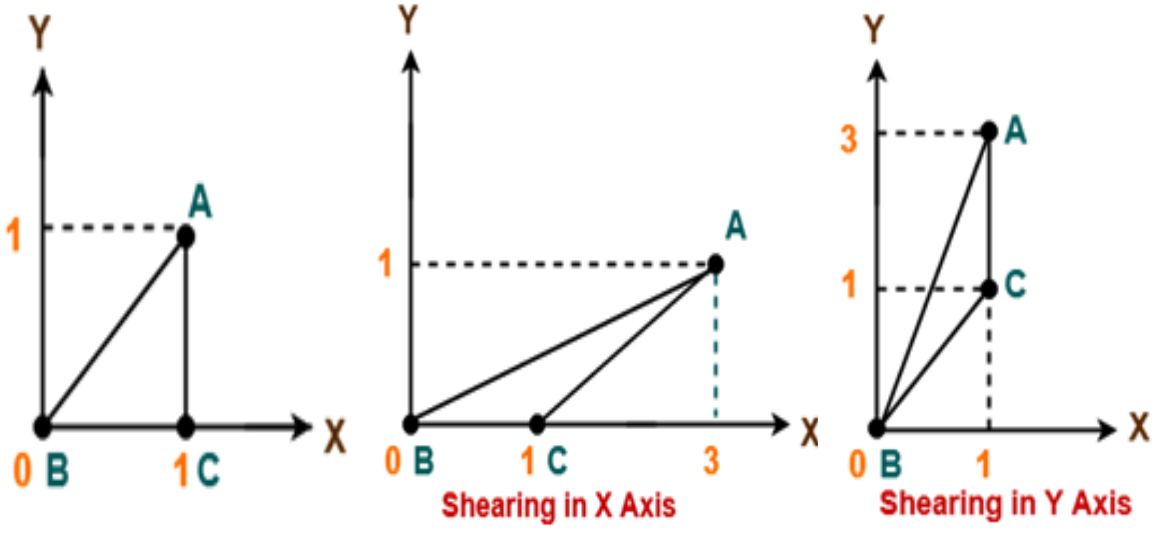
Thus, New coordinates of corner A after shearing = (1, 3).
For Coordinates B(0, 0)
Let the new coordinates of corner B after shearing = (X_{new}, Y_{new}).
Applying the shearing equations, we have-

- $X_{new} = X_{old} = 0$
- $Y_{new} = Y_{old} + Sh_y \times X_{old} = 0 + 2 \times 0 = 0$

Thus, New coordinates of corner B after shearing = (0, 0).
For Coordinates C(1, 0)
Let the new coordinates of corner C after shearing = (X_{new}, Y_{new}).
Applying the shearing equations, we have-

- $X_{new} = X_{old} = 1$
- $Y_{new} = Y_{old} + Sh_y \times X_{old} = 0 + 2 \times 1 = 2$

Thus, New coordinates of corner C after shearing = (1, 2).
Thus, New coordinates of the triangle after shearing in Y axis = A (1, 3), B(0, 0), C(1, 2).



7. COMPOSITE TRANSFORMATIONS

7.1 Need of homogenous coordinates

Sometimes we need to perform a sequence of transformations on an object like we need to scale it, then translate it and then rotate it and so on. When we perform a sequence of transformations on a single object it is called composite transformation. There can be two approaches for performing composite transformations. The first one is that we perform the transformation on objects one by one and calculate intermediate points. Another approach could be that we create a composite transformation by multiplying all the transformations and then we can directly apply it to the original coordinates and get final coordinates directly from it. The second approach is more efficient than the first one.

However, there are two types of matrices. One is a translation which is additive and the others are scaling and rotation that are multiplicative. We can combine the multiplicative and additive terms for two-dimensional geometric transformations into a single matrix representation. For that we have to convert the 2X2 matrix representations to a 3X3 matrix so that we can represent Transformation equations of all the basic geometric transformations in matrix form and can be performed using multiplications only. We will express all of them i.e. translation, rotation, scaling, reflection, shearing by a 3X3 matrix. All this is done to provide uniformity or for generalization of operations performed on the object. These uniformly represented matrices are called homogeneous matrices. Following are the homogeneous matrices representations for all the operations.

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

Translation Matrix

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

Rotation Matrix

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

Scaling Matrix

7.2 Object Matrix Representation

Moreover, if we are expressing transformations as a 3X3 matrix. Then we have to represent points or coordinate positions of the object as a matrix also. Also, we know that for matrix multiplication, the number of columns of the first matrix must be equal to the number of rows of the second matrix. That is, it is compulsory that the object matrix must have 3 rows. So, We represent each Cartesian coordinate position (x, y) also with the homogeneous coordinate (x_h, y_h, h). Here h is any non zero value which we take as 1 for simplicity. If there is a point (2,3) in Cartesian coordinates, we will represent it as (2,3,1) in homogeneous coordinates.

$$\underset{\text{Point}}{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}} \quad \underset{\text{Line}}{\begin{bmatrix} x1 & x2 \\ y1 & y2 \\ 1 & 1 \end{bmatrix}} \quad \underset{\text{Traingle}}{\begin{bmatrix} x1 & x2 & x3 \\ y1 & y2 & y3 \\ 1 & 1 & 1 \end{bmatrix}} \quad \underset{\text{Rectangle}}{\begin{bmatrix} x1 & x2 & x3 & x4 \\ y1 & y2 & y3 & y4 \\ 1 & 1 & 1 & 1 \end{bmatrix}}$$

7.3 Properties of Transformation Operations

7.3.1 Translation is additive

If we apply two successive Translations then it is Composite Translation.

$$P' = \{ T1. (T2 . P)\}$$

Suppose T1 is a translation with translation factors as tx1, ty1. While, T2 is a translation with translation factors as tx2, ty2.

$$P' = \begin{bmatrix} 1 & 0 & tx1 \\ 0 & 1 & ty1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & tx2 \\ 0 & 1 & ty2 \\ 0 & 0 & 1 \end{bmatrix} . P$$

$$P' = \begin{bmatrix} 1 & 0 & tx1+tx2 \\ 0 & 1 & ty1+ty2 \\ 0 & 0 & 1 \end{bmatrix} . P$$

So, we can say that Composite Translations are Additive. That is moving an object 10 units right and then moving it 5 units right is the same as moving an object 15 units right.

7.3.2 Rotation is additive

Two successive Rotations can be applied on a single object.

$$P' = R1.\ \{R2\ .(P)\}$$

$$P' = \begin{bmatrix} \cos\theta 1 & -\sin\theta 1 & 0 \\ \sin\theta 1 & \cos\theta 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} \cos\theta 2 & -\sin\theta 2 & 0 \\ \sin\theta 2 & \cos\theta 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} . P$$

$$P' = \begin{bmatrix} \cos\theta 1.\cos\theta 2 - \sin\theta 2 \sin\theta 1 & -\cos\theta 2 \sin\theta 1 - \cos\theta 1 \sin\theta 2 & 0 \\ \cos\theta 1.\sin\theta 2 + \cos\theta 2 \sin\theta 1 & -\sin\theta 1.\sin\theta 2 + \cos\theta 2 \cos\theta 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} . P$$

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

Therefore we can say that two successive rotations are also additive. So, rotating an object 30 degrees anticlockwise and then rotating it again 60 degrees anticlockwise is the same as rotating the object 90 degrees anticlockwise.

7.3.3 Scaling is multiplicative

While composite translation and composite rotation are additive, composite scaling is Multiplicative. We can apply two successive scaling transformations on one object as follows. Suppose we say magnify the object 2 times then magnify it 3 times. It is equivalent to saying magnify the object 6 times.

$$P' = \begin{bmatrix} sx2 & 0 & 0 \\ 0 & sy2 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} sx1 & 0 & 0 \\ 0 & sy1 & 0 \\ 0 & 0 & 1 \end{bmatrix} . P$$

$$P' = \begin{bmatrix} sx2.sx1 & 0 & 0 \\ 0 & sy2.sy1 & 0 \\ 0 & 0 & 1 \end{bmatrix} . P$$

7.4 Composite Transformation of various Type

We can apply various types of transformations on a single object. Just keep two things in mind as given below:

- Use 3X3 standard matrices for all the transformations and homogeneous coordinates for the object.
- Apply the transformations from right to left according to the sequence given. (Object matrix will be the right-most)

The first step is required because we want a uniform system for all the transformations. While the second step is required since matrix multiplication is not commutative. Hence the arrangement of matrices should be correct. We will understand it with an example of Composite transformation for Scaling and Rotation each.

Example: There is a triangle ABC A (0 , 0), B (1, 1), C (5, 2). Scale the image twice as large. Then translate it one unit to the left.

Solution:

First of all, we will make Object matrix, Scaling matrix, Translation matrix according to the values given in the question. Since we are translating the object in left direction. So t_x will be -1.

The object matrix

$$O = \begin{bmatrix} x1 & x2 & x3 \\ y1 & y2 & y3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

The scaling matrix

$$S(sx, sy) = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The translation matrix

$$T(tx, ty) = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now we will multiply matrices after arranging them in sequence.

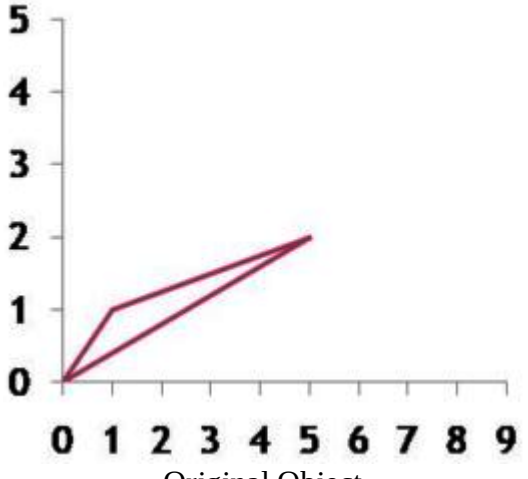
$$O' = T^* \{ S * O \}$$

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

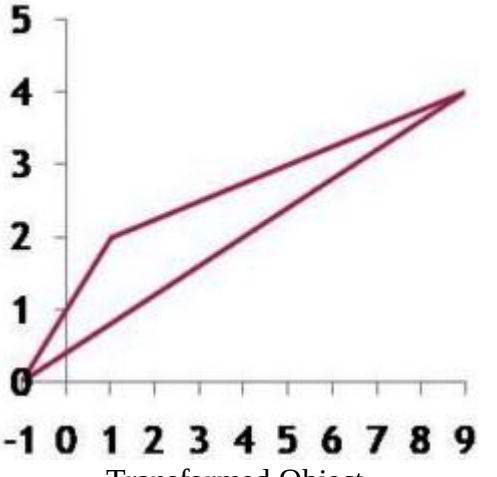
$$O' = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$O' = \begin{bmatrix} -1 & 1 & 9 \\ 0 & 2 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

Finally, the transformed object is given below with new coordinates as A'(-1, 0), B'(1, 2) and C'(9, 4).



Original Object



Transformed Object

7.4.1 Pivot Point (Fixed Point) Scaling

The Scaling that we were performing up till now is actually 2D Scaling about the origin. We can also perform Fixed-Point i.e. pivot point scaling. That is scaling about a point other than the origin. Fixed-Point Scaling can be performed easily with the help of a sequence of operations we discussed. Suppose the Fixed-Point is (xf, yf).

1. Translate the whole object such that the fixed point coincides with the origin. That is you have to shift each and every vertex of the object by the factor xf, yf. If we are moving it towards the left then the sign will be negative. So, we can express it as:

$$T = \begin{bmatrix} 1 & 0 & tx1 \\ 0 & 1 & ty1 \\ 0 & 0 & 1 \end{bmatrix} \quad O' = \begin{bmatrix} 1 & 0 & -xf \\ 0 & 1 & -yf \\ 0 & 0 & 1 \end{bmatrix} .O$$

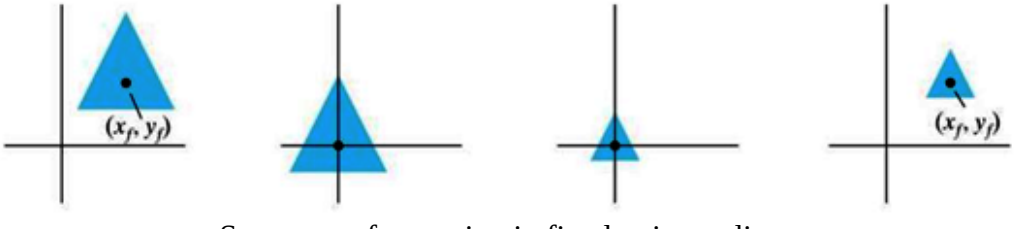
2. Now, we will perform scaling about origin. And remember that origin and fixed-point are actually same at this instant. Since we have shifted the whole object.

$$O'' = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} .O'$$

3. We cannot keep the object where it is. We will shift it back to its original position. If we are moving it towards right then translation factors will be positive.

$$O''' = \begin{bmatrix} 1 & 0 & xf \\ 0 & 1 & yf \\ 0 & 0 & 1 \end{bmatrix} .O''$$

We can arrange all these matrices and then we can perform the composite transformation. Remember to perform all the transformations from right to left. Left-most being the object matrix. Also, remember Matrix multiplication is associative but not commutative.



Sequence of operation in fixed point scaling

$$O' = \overset{3}{T(x_f, y_f)} \cdot \overset{2}{S(s_x, s_y)} \cdot \overset{1}{T(-x_f, -y_f)} O$$

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

This fixed-point may be inside the object which we are going to scale. It may be outside also. We can also derive a composite matrix by multiplying all these matrices and then we just have to put the values in that single matrix only.

7.4.2 Pivot Point (Fixed Point) Rotation

We can generate rotation about any given point other than the origin. This is Pivot Point Rotation. For 2D rotation, we must have an angle of rotation, the point about which rotation is performed. We must also know the direction of rotation. Whether it is clockwise or anti-clockwise. For anticlockwise angle is taken as positive. While for clockwise it is taken as negative. Now, the pivot point can be the origin or some other point inside or outside the object. If the pivot point is the origin it is simple 2D rotation. But if it is some other point. Then it is Fixed point Rotation. We can perform Fixed point rotation with the help of composite transformation. We will use a sequence of operations again. First, we will move the given point to the origin. Then, we will perform a rotation about the origin. Finally, we will shift the object back to its original position. It can be summarized as Translate-Rotate-Translate.

1. Translate the object so that given rotation point coincides with origin. Suppose the given point is (x_r, y_r). Then translation factors will be $(t_x, t_y) = (-x_r, -y_r)$. Since we are moving the point downwards and right.
2. Rotate the object about origin with the given angle.
3. Translate the object back to its original position. In this case translation factors will be positive. Since we are moving the point upwards and towards left. That is $(t_x, t_y) = (x_r, y_r)$.

Sequence of operations in fixed point rotation

$$O' = T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r).O$$

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

EXERCISE

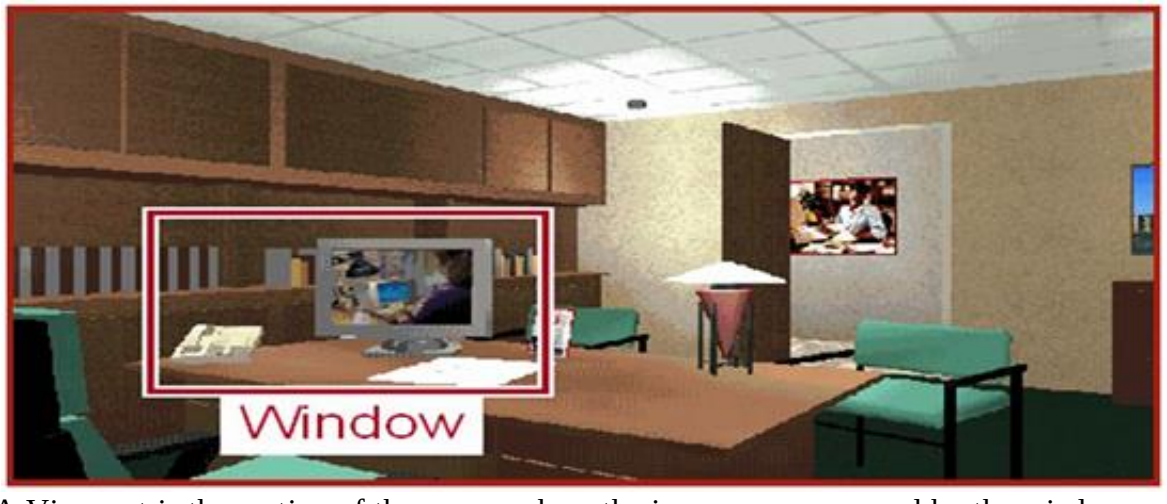
1. Given a square with coordinate points A(0, 3), B(3, 3), C(3, 0), D(0, 0). Apply the translation with distance 1 towards X axis and 1 towards Y axis. Obtain the new coordinates of the square.
2. Prove that 2D rotations about the origin are commutative i.e. $R_1 R_2 = R_2 R_1$.
3. Consider a triangle ABC A(2,2) B(3,4) C(4,2) .Scale the triangle twice as large about the origin.
4. Given a triangle with coordinate points A (3, 4), B (6, 4), C (5, 6). Apply the reflection on the X-axis, Y-axis and Z-axis. Obtain the new coordinates of the object.
5. There is a triangle ABC A(4,6) B(2,2)C(6,2). Scale the image thrice as large about the point (4,4).

4. 2D VIEWING & CLIPPING

All objects in the real world have size. We use a unit of measure to describe both the size of an object as well as the location of the object in the real world. For example, meters can be used to specify both size and distance. When we want to display an image of an object on the screen, we have to use a screen coordinate system that defines the location of the object with reference to the relative position as in the real world. Once, the screen coordinate system is selected, the picture is scaled down to display it on the screen.

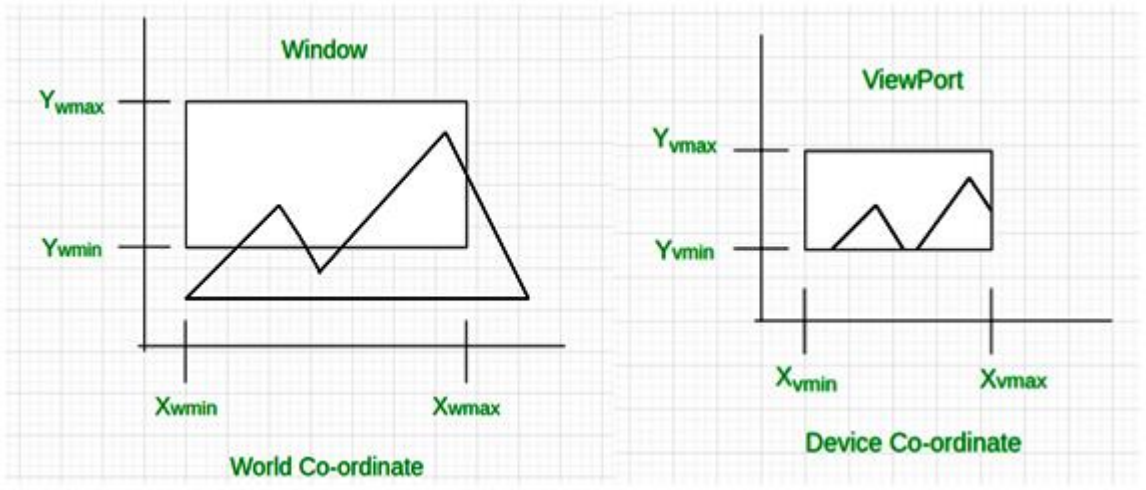
1. WINDOWING AND CLIPPING

The world coordinate system is used to define the position of objects in the real world. This system does not depend on the screen coordinate system, so the interval of number can be anything (positive, negative or decimal). Sometimes the complete picture of object in the world coordinate system is too large and complicate to clearly show on the screen hence we need to show only some part of the object.This is carried out by windowing and clipping. A Window is a rectangular region in the world coordinate system that defines what is to be viewedwhere as viewport decide where it is to be viewed.



A Viewport is the section of the screen where the images encompassed by the window on the world coordinate system will be drawn or displayed. A coordinate transformation from world coordinate system to viewport coordinates (device coordinates) is required to display the image on the window. The viewport uses the screen coordinate system. So, effectively, this transformation is from the world coordinate system to the screen coordinate system.

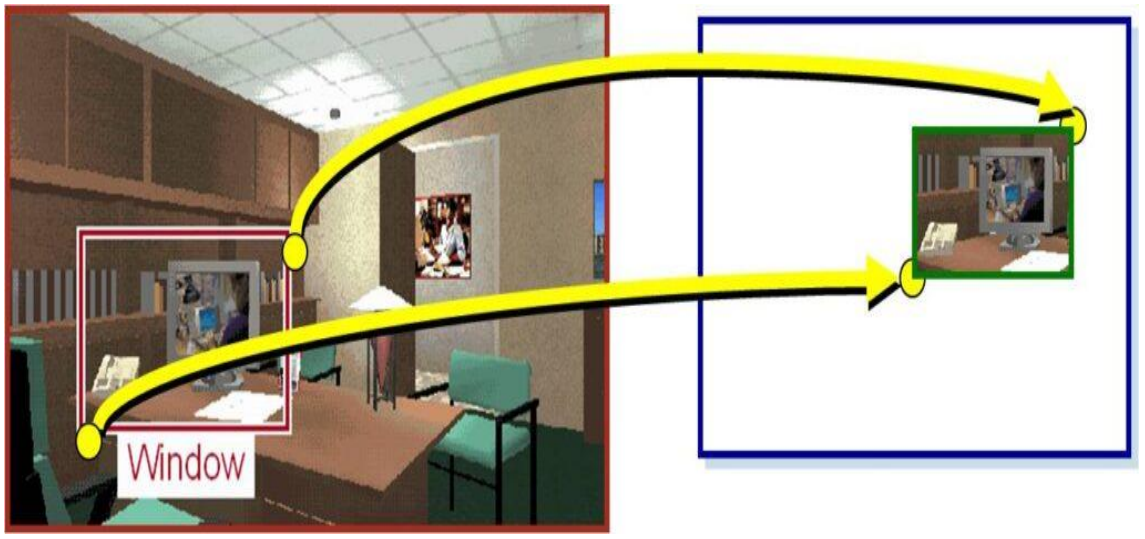
When a window is placed on the real world scenes, only certain objects and parts of objects can be seen. Points and lines which are outside the window are cut off from view. This process of cutting off the parts of the image of the world is called Clipping. In clipping, we examine each line to determine whether or not it is completely inside the window, completely outside the window, or crosses a window boundary. If it is inside the window, the line is displayed. If it is outside the window, the lines and points are not displayed. If a line crosses the boundary, we must determine the point of intersection and display only the part which lies inside the window.



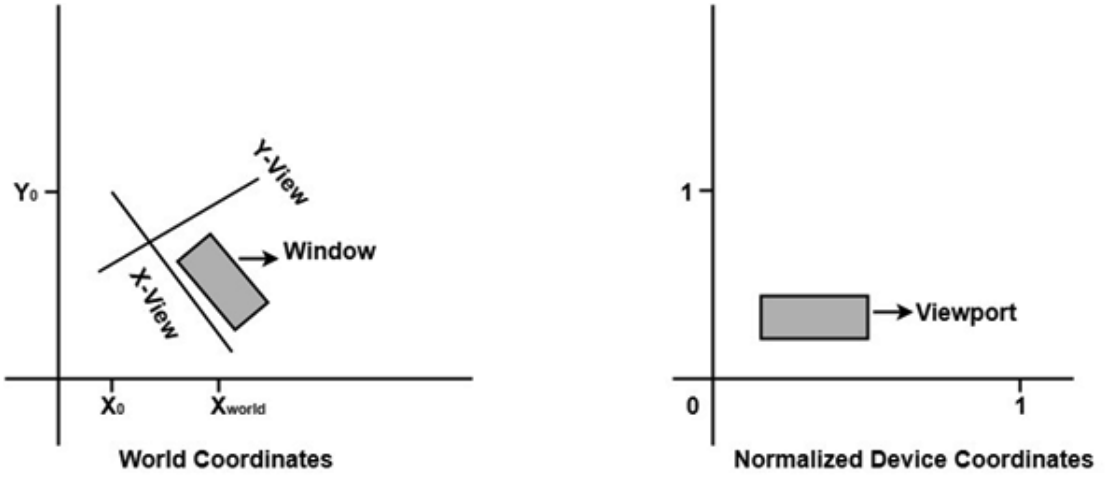
Window to Viewport Transformation is the transformation from world to device coordinates. It is actually a combination of many transformations. It basically involves translation, scaling, rotation transformations. It also requires methods to delete those portions of the scene which are outside the selected display area.

1.1 Window to Viewport Transformation

As discussed above, there are two coordinate systems. One is the World Coordinate system while other is the Device Coordinate system. Therefore, The mapping of a world coordinate scene to device coordinate is Window to Viewport Transformation or Windowing transformation.



1.2 Normalized Coordinates

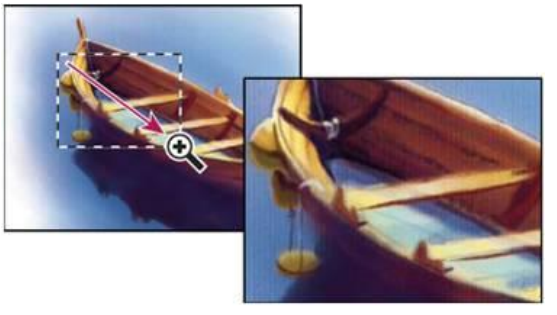


Viewports are defined within the unit square called normalized coordinates. Normalized coordinates are useful for separating viewing and other transformations from specific output device requirements so that the graphics package remains largely device-independent.

1.3 Applications of Viewport

We can view the objects at different positions on the display area of an output device by changing the positions of the viewport. Therefore, We can change the size and proportions of displayed objects by varying the size of the viewport. It is also possible to determine many viewports on different areas of display and view the same object at a different angle in each viewport.

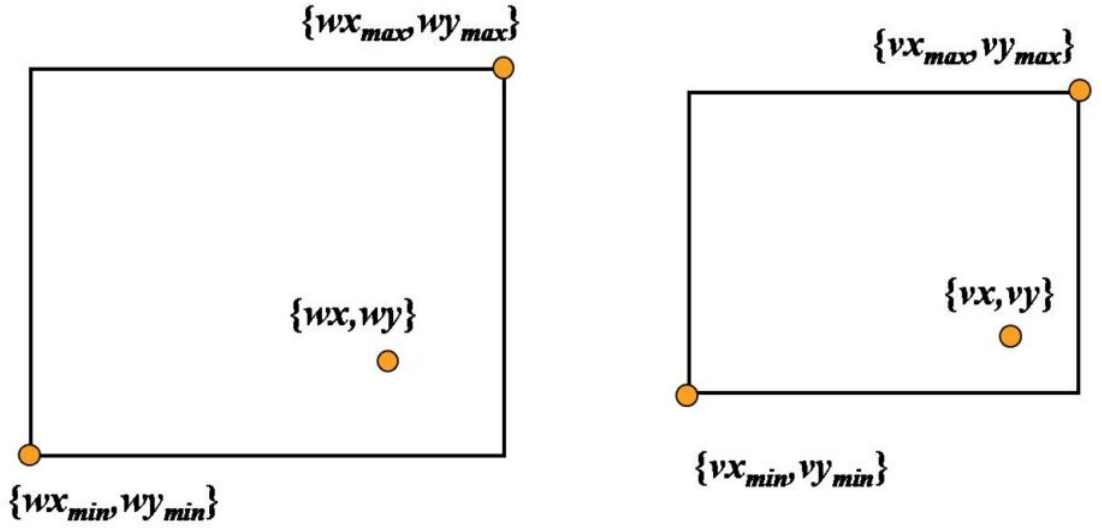
Zooming Effect: Zooming Effect is Successively mapping different sized windows on a fixed size viewport. As the windows are made smaller, so we zoom-in on some part of a scene to view details that were not visible in large windows. Also, We can obtain more overview by zooming-out from a section of a scene with successively larger windows.



Panning Effect: Panning Effect is actually moving a fixed size window across various objects in a scene.

1.4 Window to viewport Transformation

A window is specified by four world coordinates, Wx_{max}, Wx_{min}, Wy_{max}, Wy_{min}. While, A viewport is specified by Vx_{max}, Vx_{min}, Vy_{max}, Vy_{min}. Moreover, The objective of window to viewport mapping is to convert the world coordinate (W_x,W_y) of an arbitrary point to its corresponding normalized device coordinate (V_x,V_y).



Viewport Transformation: We need have to map a point at the position W_x, W_y in the window into the position V_x, V_y in the viewport.
Window to Viewport Mapping: In order to maintain the same relative placement of the point in the viewport as in window, we require

$$\frac{wx - wx_{min}}{vx - vx_{min}} = \frac{wx_{max} - wx_{min}}{vx_{max} - vx_{min}} \quad and \quad \frac{wy - wy_{min}}{vy - vy_{min}} = \frac{wy_{max} - wy_{min}}{vy_{max} - vy_{min}}$$

Solving for Vx, we get

$$vx = (wx - wx_{min}) * \frac{vx_{max} - vx_{min}}{wx_{max} - wx_{min}} + vx_{min}$$

Similarly, solving for Vy, we get

$$vy = (wy - wy_{min}) * \frac{vy_{max} - vy_{min}}{wy_{max} - wy_{min}} + vy_{min}$$

Algorithm

Step1: Translate window to origin by $T_x = -Wx_{min}$ $T_y = -Wy_{min}$

Step2: Then, Scaling of the window to match its size to the viewport

$$S_x = (vx_{max} - vx_{min}) / (wx_{max} - wx_{min})$$

$$S_y = (vy_{max} - vy_{min}) / (wy_{max} - wy_{min})$$

Step3:Again translate viewport to its correct position on screen by

$$T_x = Vx_{min}$$

$$T_y = Vy_{min}$$

We can represent above three steps in matrix form as,

$$V_T = T1 * S * T$$

T = Translate window to the origin

S=Scaling of the window to viewport size

T1=Translating viewport on screen.

$$N = \begin{bmatrix} 1 & 0 & vx_{min} \\ 0 & 1 & vy_{min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -wx_{min} \\ 0 & 1 & -wy_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} vx \\ vy \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & vx_{min} \\ 0 & 1 & vy_{min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -wx_{min} \\ 0 & 1 & -wy_{min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} wx \\ wy \\ 1 \end{bmatrix}$$

Example: Find normalization transformation that maps a window whose lower-left corner is at (1,1) and upper right corner is at (3,5) onto: a)Viewport with lower-left corner (0,0) and upper right corner (1,1) b) Viewport with lower left corner (0,0) and upper right corner (1/2,1/2).

Solution:

$Wx_{min}=1$, $Wy_{min}=1$, $Wx_{max}=3$, $Wy_{max}=5$

a) $Vx_{min}=0$, $Vy_{min}=0$, $Vx_{max}=1$, $Vy_{max}=1$

b) $Vx_{min}=0$, $Vy_{min}=0$, $Vx_{max}=1/2$, $Vy_{max}=1/2$

a) $S_x = (Vx_{max} - Vx_{min}) / (Wx_{max} - Wx_{min})$, $S_y = (Vy_{max} - Vy_{min}) / (Wy_{max} - Wy_{min})$

While, Putting the values:

$S_x = 1\text{-}0/3\text{-}1 = 1/2$,

$S_y = 1\text{-}0/5\text{-}1 = 1/4$

$$N=\begin{bmatrix}1 & 0 & vx_{min}\\ 0 & 1 & vy_{min}\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}sx & 0 & 0\\ 0 & sy & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & -wx_{min}\\ 0 & 1 & -wy_{min}\\ 0 & 0 & 1\end{bmatrix}$$

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

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

b) $S_x = (Vx_{max} - Vx_{min}) / (Wx_{max} - Wx_{min})$, $S_y = (Vy_{max} - Vy_{min}) / (Wy_{max} - Wy_{min})$

While, Putting the values:

$S_x = (1/2\text{-}0)/3\text{-}1 = (1/2)/2 = 1/4$

$S_y = (1/2\text{-}0)/5\text{-}1 = (1/2)/4 = 1/8$

$$N=\begin{bmatrix}1 & 0 & vx_{min}\\ 0 & 1 & vy_{min}\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}sx & 0 & 0\\ 0 & sy & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & -wx_{min}\\ 0 & 1 & -wy_{min}\\ 0 & 0 & 1\end{bmatrix}$$

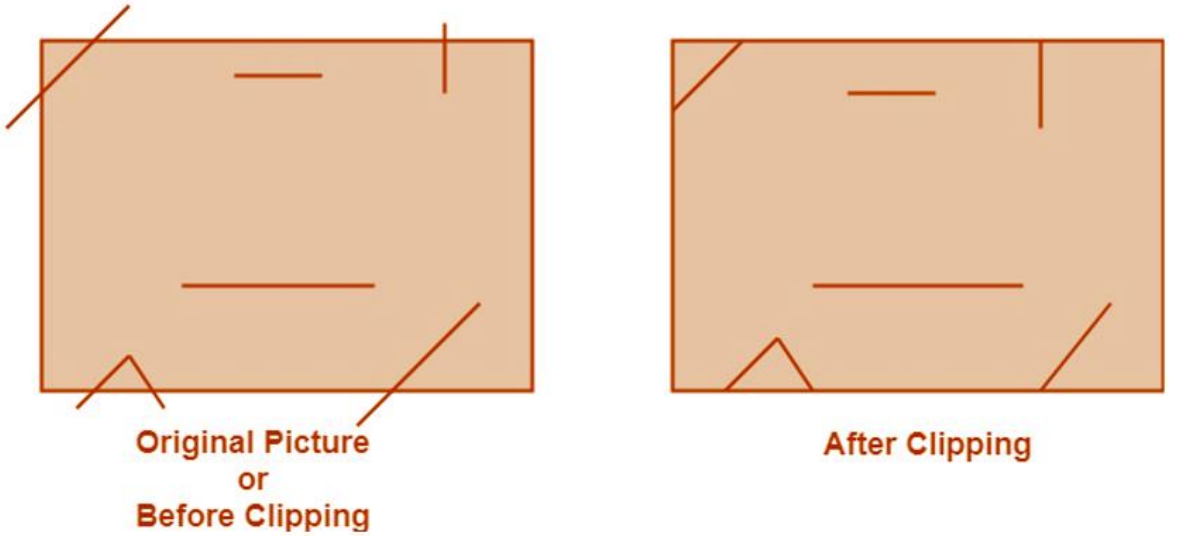
$$N = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/4 & 0 & 0 \\ 0 & 1/8 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$N = \begin{bmatrix} 1/4 & 0 & -1/4 \\ 0 & 1/8 & -1/8 \\ 0 & 0 & 1 \end{bmatrix}$$

2. CLIPPING

2.1 Clipping and Clip Window

We defined what is clipping in first section. Now, we will discuss how clipping can be performed. First of all we have to define a region, called Clip Window. Then, we will keep all the portions or objects which are inside this window while all the things outside this window are discarded. So clipping involves identifying which portion is outside the clip window and discarding that outer portion. Clip window may be rectangular or any other polygon or it may be a curve also. The following picture shows the clipping effect.



2.2 Applications of Clipping

1. We can select a part from a scene for displaying just like we use camera to click a picture. In camera, we focus at a particular portion and then discard the things which are out of frame.

2. In three-dimensional views, only some surfaces are visible and others are hidden. We can use clipping to identify visible surfaces.
3. During the process of Rasterization, we see aliasing effect (zig-zag) in line segments or object boundaries. We can use clipping for antialiasing.
4. Solid-modeling procedures also use Clipping.
5. We can use clipping for multiwindow environment
6. With the help of Clipping Operation, we can select a part of picture, copy, move or delete that part. Various painting and Drawing Applications have this feature.

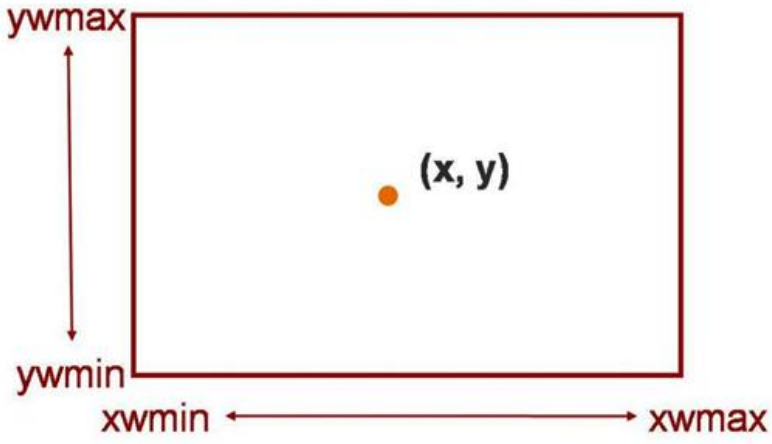
2.3 Approaches of Clipping

Clipping can be performed in two ways:We can perform Clipping in world coordinates before mapping it to device coordinates so that we can save unnecessary operations. In this, we need not to map those objects that we will ultimately discard. However, in viewport clipping we perform clipping after mapping it to device coordinates.Various types of Clipping are Point Clipping, Line Clipping, Polygon Clipping, Text Clipping and Curve Clipping.

2.3.1 Point Clipping

First of all consider a point P(x, y). We can specify Edges of Clip Window by (Xw_{min},Xw_{max},Yw_{min},Yw_{max}). If following four inequalities are not satisfied, Then point is clipped, otherwise, displayed.

$$Xw_{min} <= x <= Xw_{max}$$
$$Yw_{min} <= y <= Yw_{max}$$



2.3.2 Line Clipping

We specify a line with its end-points. There are three possible cases for a line that we need to consider:

- First, will check if a line is completely inside the window. If it is, then we will display it completely.
- Otherwise, we will check if a line is completely outside the window. If it is, then we will discard that line.

- Thirdly, if the line is neither completely inside nor completely outside i.e. it is partially visible. It is to be clipped.

The following sections 3 and 4 discusses line clipping algorithms.

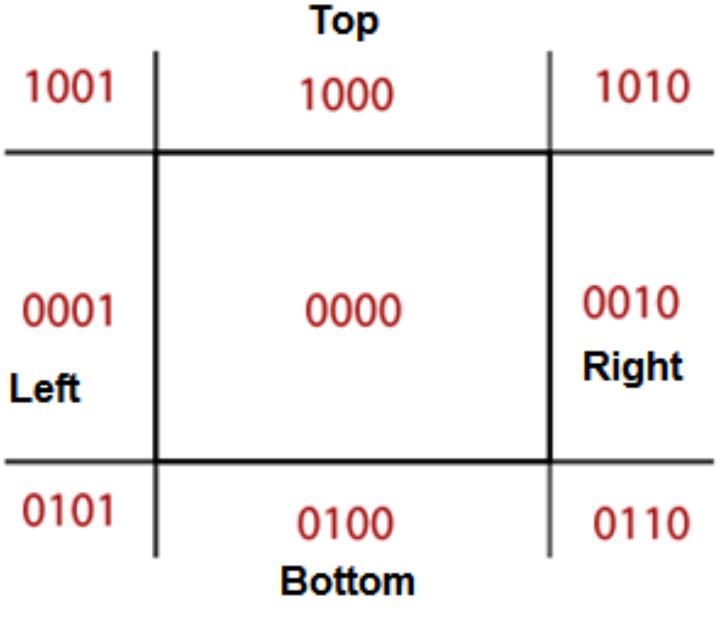
3. COHEN SUTHERLAND LINE CLIPPING ALGORITHM

Cohen Sutherland Algorithm is a line clipping algorithm that cuts lines to portions which are within a rectangular area. It eliminates the lines from a given set of lines and rectangle view port which belongs outside the area of interest and clip those lines which are partially inside the viewport.In this algorithm, first of all, it is detected whether line lies inside the screen or it is outside the screen. All lines come under any one of the following categories:

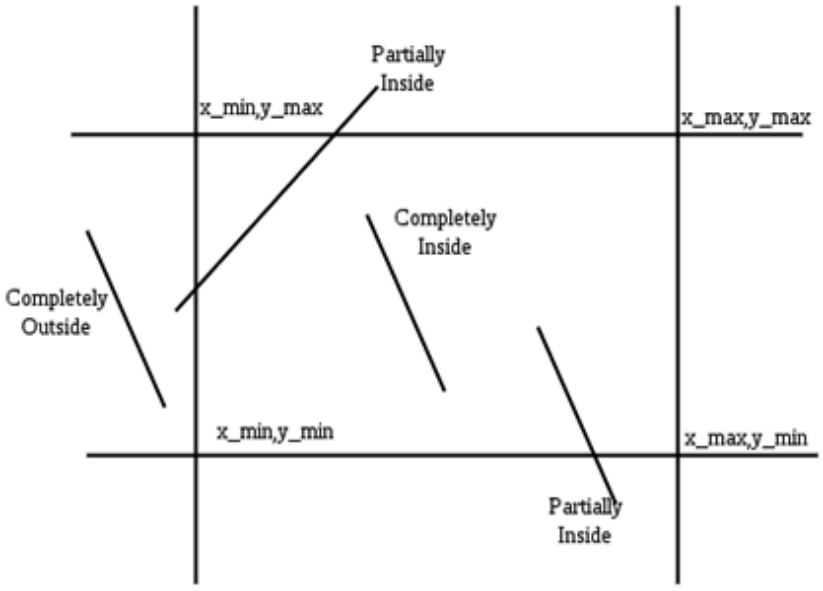
1. Visible: If a line lies within the window, i.e., both endpoints of the line lies within the window, a line is visible and will be displayed as it is.
2. Not Visible: If a line lies outside the window it will be invisible and rejected. Such lines will not be displayed. If any one of the following inequalities is satisfied, then the line is considered invisible. Let A (x_1,y_2) and B (x_2,y_2) are endpoints of line and (Xw_{min}, Yw_{min}) (Xw_{max}, Yw_{max}) are coordinates of the window.

$$x_1>Xw_{max} \text{ and } x_2>Xw_{max} \text{ and } y_1>Yw_{max} \text{ and } y_2>Yw_{max}$$
$$x_1<Xw_{min} \text{ and } x_2<Xw_{min} \text{ and } y_1<Yw_{min} \text{ and } y_2<Yw_{min}$$

3. Clipping Case: If the line is neither visible case nor invisible case. It is considered to be clipped case. In this, the category of a line is found based on nine regions given in figure below. All nine regions are assigned unique codes made up of 4 bits. If both endpoints of the line have end bits zero, then the line is considered to be visible. The allocation of bits depends on "TBRL" (Top, Bottom, Right, Left) rule.



The center area is having the code, 0000, i.e., region 5 is considered a rectangle window. Following figure show lines of various types.



Algorithm

Step 1:Calculate positions of both endpoints of the line
Step 2:Perform OR operation on both of these end-points

Step 3: If the OR operation gives 0000 then
line is considered to be visible
else
Perform AND operation on both endpoints
If AND = 0000 then
the line is considered for the clipping
else AND ≠ 0000
the line is invisible

Step 4:If a line is clipped case, find an intersection with boundaries of the window using slop equation

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

(a) If L bit "1" line intersects with left boundary of window

$$y_c=y_1 + m(Xw_{min} - x_1)$$

(b) If R bit is "1" line intersect with right boundary of window

$$y_c=y_1 + m(Xw_{max} - x_1)$$

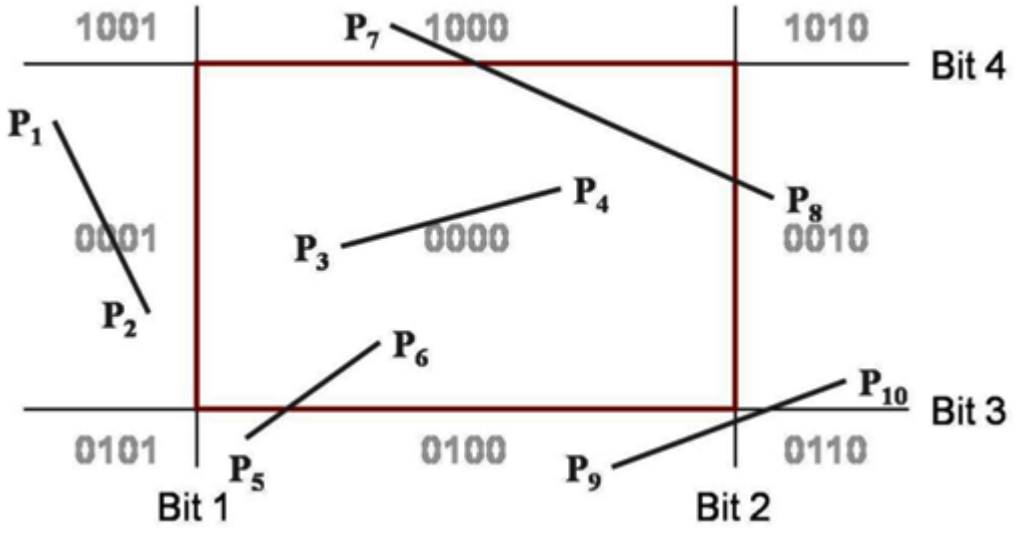
(c) If B bit is "1" line intersects with bottom boundary of window

$$x_c=x_1 + (Yw_{min} - y_1)/m$$

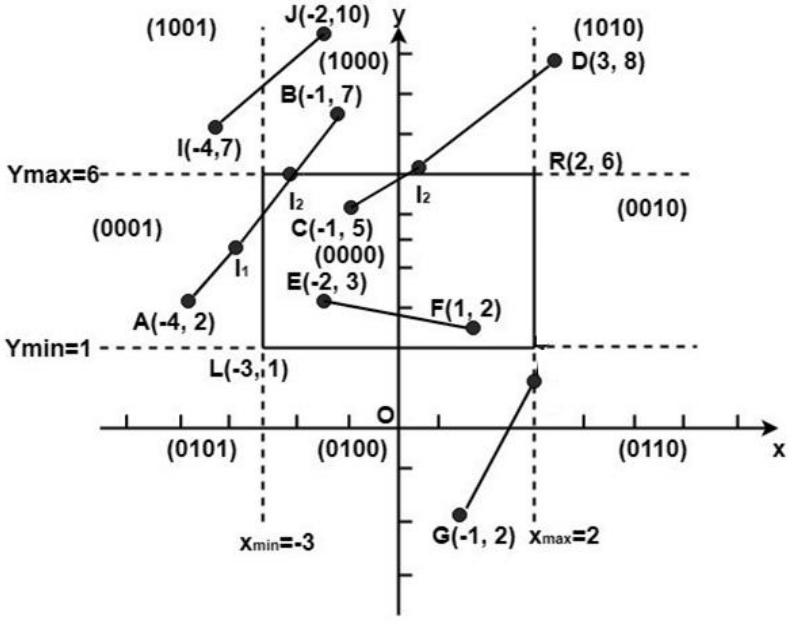
(d) If T bit is "1" line intersects with the top boundary of window

$$x_c=x_1 + (Yw_{max} - y_1)/m$$

Special Case: Compute intersections with window boundary for lines that can't be classified quickly (here, line P9P10 for which the AND operation will give result 0000 yet it is not candidate for clipping. So find out the intersections with bottom and right window boundary. If it is not found then line is invisible.)



Example: Let R be the rectangular window whose lower left-hand corner is at L (-3, 1) and upper right-hand corner is at R (2, 6). Find the region codes for the endpoints in figure.



Solution:
So, as per the region codes,

A(-4,2)→0001	F(1,2)→0000
B(-1,7)→1000	G(1,-2)→0100
C(-1,5)→0000	H(3,3)→0100
D(3,8)→1010	I(-4,7)→1001
E (-2,3)→0000	J(-2,10)→1000

We place the line segments in their appropriate categories by testing the region codes found in the problem.
Case 1 (visible): EF since the region code for both endpoints is 0000.
Case 2 (not visible): IJ since (1001) AND (1000)=1000 (which is not 0000). GH since (0100) AND (0100)=0100 (which is not 0000).
Case 3 (candidate for clipping): AB since (0001) AND (1000)=0000, CD since (0000) AND (1010)=0000.
Finding intersection points for y for left and x for top boundary of AB, we get, $(-3, {}^{11}/_{3})$ and $(-{}^{8}/_{5}, 6)$ as end points.
For CD, C is inside we have to just calculate intersection points with top boundary. By using the formula, we get $({}^{3}/_{5}, 6)$

4. LIANG-BARSKY LINE CLIPPING ALGORITHM

The algorithm was introduced by You-Dong Liang and Brian A. Barsky. It is also used for line clipping and is a more powerful algorithm than the Cohen-Sutherland algorithm.

- We can use the parametric equation of line and inequalities.
- These are used to describe the range of windows to find out the intersection points between the line and the clipping window.
- The parametric line is also known as "Cyrus-Beck."

In this algorithm, we have to find the intersection point based on a time interval.
Time interval (t) can be defined as travelling time between initial position (0) to final position (1). Sowe have,

$$0 < t < 1$$

We have the formula to find x and y points of the line,

$$x = x_1 + t\Delta x \text{ (For point x)}$$
$$y = y_1 + t\Delta y \text{ (For point y)}$$

To check that the point lies between the window or outside the conditions can be given as,

$$x_1 + t\Delta x >= Xw_{min}$$
$$x_1 + t\Delta x <= Xw_{max}$$
$$y_1 + t\Delta y >= Yw_{min}$$
$$y_1 + t\Delta y <= Yw_{max}$$

We can take a common expression for above four conditions. It will be,

$$t.p_k <= q_k \text{ (Here the value of k is multiple)}$$

So,

$$t = q_k / p_k$$

$p_1 = -\Delta x$ $q_1 = x_1 - Xw_{min}$ (For left boundary)

$p_2 = \Delta x$ $q_2 = Xw_{max} - x_1$ (For right boundary)

$p_3 = -\Delta y$ $q_3 = y_1 - Yw_{min}$ (For bottom boundary)

$p_4 = \Delta y$ $q_4 = Yw_{max} - y_1$ (For top Boundary)

Algorithm

Step 1: Set the endpoints of the line (x_1, y_1) and (x_2, y_2).
Step 2: Calculate the value of p_1, p_2,p_3, p_4 and q_1, q_2, q_3,q_4.
Step 3: Now we calculate the value of t
$t_1 = 0$ (For initial point)
$t_2 = 1$ (For final point)
Step 4: Now, we have to calculate the value of p_k and q_k

If $p_k = 0$then
{The line is parallel to the window}
If $q_k < 0$then
{The line is completely outside the window}

Step 5: If we have non zero value of p_k,

If $p_k < 0$then
$t_1 = \max(0, q_k / p_k)$
If $p_k > 0$then
$t_2 = \min(1, q_k / p_k)$
if $t_1 < t_2$
{
If t_1 value is changed then
the first point is outside the window.
If t_2 value is changed then
the second point is outside the window
}
else if $t_1 > t_2$then
Line is completely outside the window

Step 6: Stop.

Example:Let a rectangular window size with (5, 9). The points of the line are (4, 12) and (8, 8). Use the Liang- Barsky algorithm to clip the line and find the intersection point.

Solution:

The initial point of the line $(p_1) = (4, 12)$
The ending point of the line $(p_2) = (8, 8)$
$x_1 = 4$, $x_2 = 8$

$y_1 = 12, y_2 = 8$

$Xw_{min} = 5, Xw_{max} = 9$

$Yw_{min} = 5, Yw_{max} = 9$

Step 1: We have to calculate the value of Δx and Δy

$$\Delta x = x_2 - x_1 = 8-4 = 4$$

$$\Delta y = y_2 - y_1 = 8-12 = -4$$

Step 2: Now, we will calculate,

$p_1 = -4 \quad q_1 = 4-5 = -1$

$p_2 = 4 \quad q_2 = 9-4 = 5$

$p_3 = 4 \quad q_3 = 12-5 = 7$

$p_4 = -4 \quad q_4 = 9-12 = -3$

Step 3: Now, we will calculate t_1 value,

If $p_1, p_4 < 0$ then,

$t_1 = \max(0, q_k/p_k)$

$= \max(0, q_1/p_1, q_4/p_4)$

$= \max(0, 1/4, 3/4)$

$t_1 = 3/4$

If $p_2, p_3 > 0$ then,

$t_2 = \min(1, q_k/p_k)$

$= \min(1, q_2/p_2, q_3/p_3)$

$= \min(1, 5/4, 7/4)$

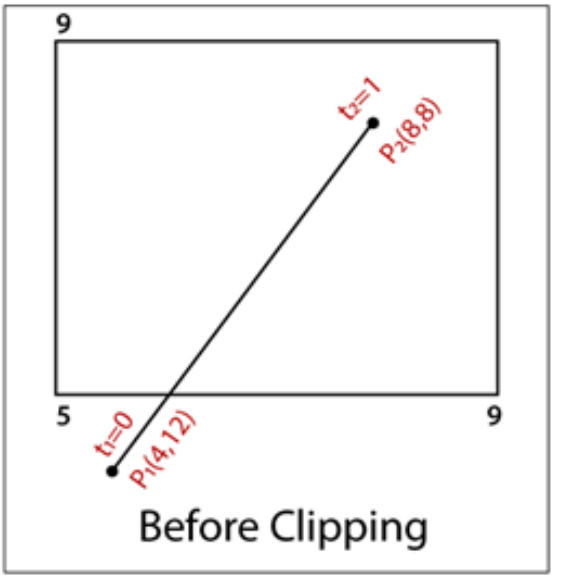
$t_2 = 1$

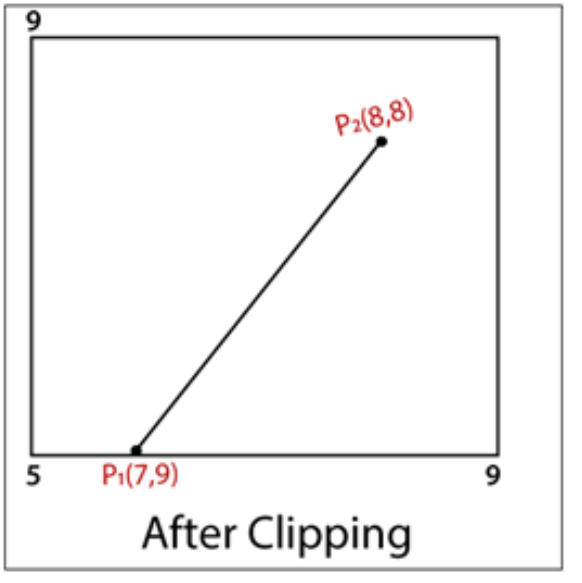
Step 4: Now, we have to calculate the intersection point.

$x = x_1 + t_1\Delta x = 4 + 3/4 * 4 = 7$

$y = y_1 + t_1\Delta y = 12 + 3/4 *(-4) = 9$

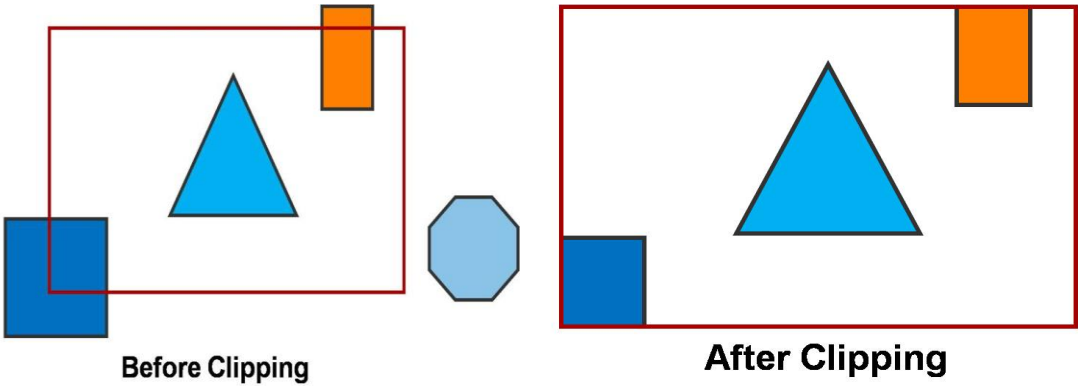
The coordinates intersection point = (7, 9)





5. POLYGON CLIPPING

Polygon clipping tells us how to clip a polygon against Clip Window. A polygon is actually made of line segments. Therefore, we can use line clipping procedures for Polygon Clipping. Though it may be used but it may result in unconnected line segments. So, we need a clipping algorithm that can create a closed polygon. There are mainly two polygon clipping algorithms available in Computer Graphics along with the Line Clipping Algorithms.



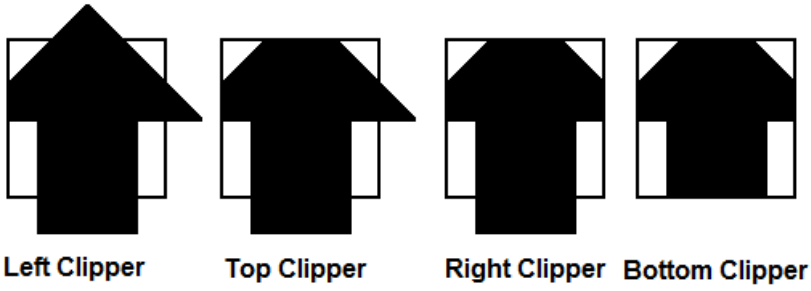
5.1 Sutherland-Hodgeman Polygon Clipping Algorithm

This algorithm was purposed by Sutherland and Hodgeman to determine which portions of the polygon were visible within a window and clipping out the portions that are outside the window.This algorithm accepts as input a series of polygon vertices, V_1, V_2, …. V_n. These vertices define polygon edges from V_i, to V_{i+1} and from V_n to V_1.

The algorithm clips a polygon by processing the polygon boundary as a whole against each window edge. It means all the vertices are processed for clipping against each individual clip boundary of clipping window one by one. Starting with the initial set of vertices, the polygon is first clipped against left window boundary and a new set of vertices is produced. This new set is used as input for clipping against top boundary. It produces another set of vertices which is used against right window boundary. The output of this is given as input to bottom window boundary and final set of vertices is produced. All the vertices are connected and the clipped polygon is generated. During processing the pairs of verticesfollowing four possible cases may arise.

1. If both vertices are outside then nothing is saved. (Because It means that edge is completely outside the clip window).
2. If both vertices are inside then we will save both. (Since it means that the edge connecting those vertices is completely inside the clip window).
3. If the first vertex is inside and the second vertex is outside then the first vertex is saved as it is and the intersection point of that polygon edge with clip window is saved.

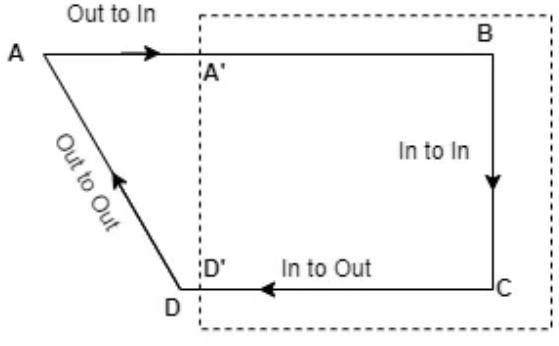
4. If the second vertex is inside and the first vertex is outside then we will save the second vertex. We will also save the intersection point of the polygon edge with a clip window.



All four cases have been explained in the following figures:

Case 1 (out → in): If the starting vertex is outside the window boundary and ending vertex is inside then we preserve or save both point of intersection of polygon edge with window boundary and ending vertex in the output vertex list.

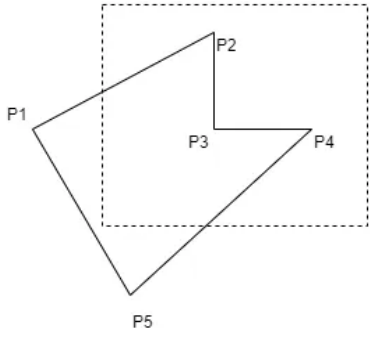
Case 2 (in → in):If both the starting and ending vertices are inside the window boundary then only the ending vertex is preserved or saved in output vertex list.
Case 3 (in → out): If the starting vertex is inside the window and ending vertex is outside then only the point of intersection of polygon edge with window boundary is saved in output list.
Case 4 (out → out): If both the starting and ending vertices are outside the window boundary then nothing is added or saved in the output vertex list.



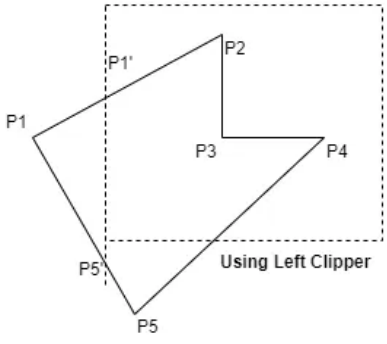
Four cases of polygon clipping

Case	Output
Out→In	A'B *[2 Vertices, Point of intersect with window and ending edge]*
In→In	C *[1 ending vertex]*
In→Out	D' *[Point of intersect with window]*
Out→Out	N/A *[Nothing is included]*

Example: Consider the following polygon with vertices P1, P2, P3, P4, P5. Inside in a viewing window. ApplySutherland-Hodgeman Polygon Clipping Algorithm on this polygon.

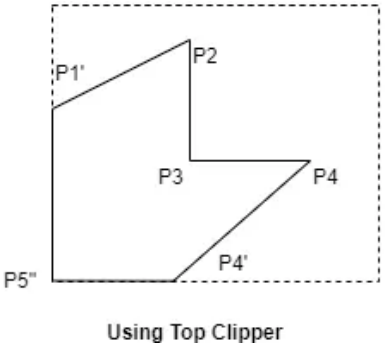


Clipping polygon engaged left window boundary: Here the input vertex list is P1, P2, P3, P4 P5 and the edges are P1P2, P2P3, P3P4, P4P5 and P5P1. Now, consider all the edge with respect to (w.r.t.) left window boundary.



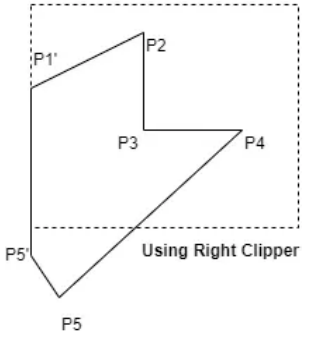
Edge	Type	Output
P1P2	Out → In	P1'P2
P2P3	In → In	P3
P3P4	In → In	P4
P4P5	In → In	P5
P5P1	In → Out	P5′

Clipping polygon engaged top window boundary: Here the input vertex list is P1′, P2, P3, P4, P4′, P5" and the edges are P1'P2, P2P3, P3P4, P4P4′, P4'P5" and P5"P1. Now, consider all the edge with respect to (w.r.t.) top window boundary. Here the input vertex list is P1′, P2, P3, P4, P5, P5′ and the edges are P1'P2, P2P3, P3P4, P4P5, P5P5′ and P5'P1′. Now, consider all the edge with respect to (w.r.t.) right window boundary. There will be no changes because all edges lies inside the top clipping window.



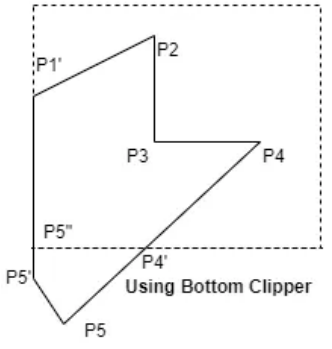
Edge	Type	Output
P1'P2	In → In	P2
P2P3	In → In	P3
P3P4	In → In	P4
P4P4′	In → In	P4′
P4'P5"	In → In	P5"
P5"P1′	In → In	P1′

Clipping polygon engaged right window boundary: Here the input vertex list is P1′, P2, P3, P4, P5, P5′ and the edges are P1’P2, P2P3, P3P4, P4P5, P5P5′ and P5’P1′. Now, consider all the edge with respect to (w.r.t.) right window boundary. There will be no changes because all edges lies inside the right clipping window.



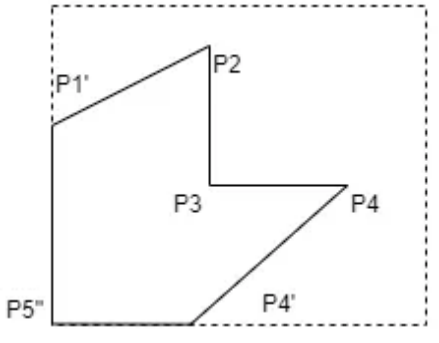
Edge	Type	Output
P1’P2	In → In	‘P2
P2P3	In → In	P3
P3P4	In → In	P4
P4P5	In → In	P5
P5P5′	In → In	P5′
P5’P1′	In → In	P1′

Clipping polygon engaged bottom window boundary: Here the input vertex list is P1′, P2, P3, P4, P5, P5′ and the edges are P1’P2, P2P3, P3P4, P4P5 and P5P5′. Now, consider all the edge with respect to (w.r.t.) bottom window boundary.

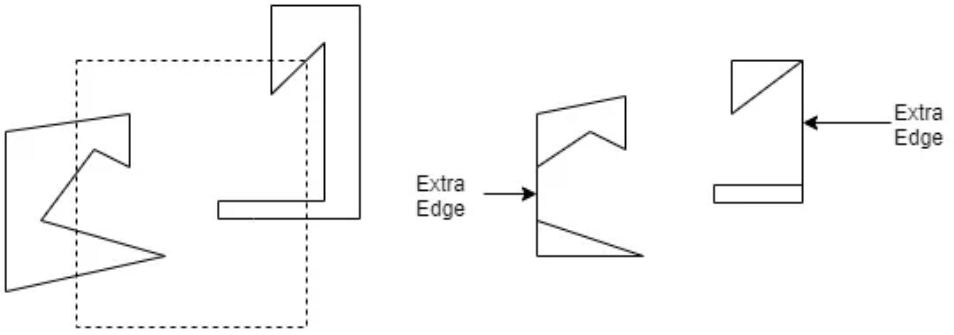


Edge	Type	Output
P1’P2	In → In	P2
P2P3	In → In	P3
P3P4	In → In	P4
P4P5	In → Out	P4′
P5P5′	Out → Out	N/A
P5’P1′	Out → In	P5”

Final output using Sutherland-Hodgeman Polygon Clipping Algorithm will be:



Using this algorithm, we can clip convex polygons correctly but we may get wrong results if we process concave polygons with it. The reason behind this is that we have a set of vertices as output and we join the last vertex with the first one to make a bounded polygon. This may produce wrong result with unwanted edge. For this, Weiler-Atherton Algorithm was developed which works for both types of polygons

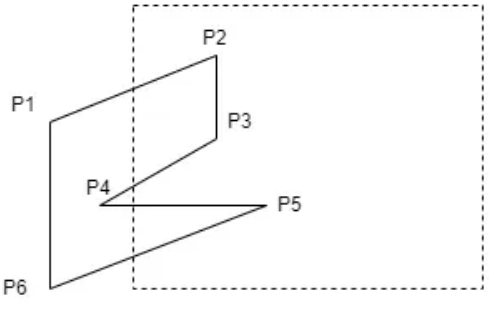


5.2 Weiler-Atherton Polygon Clipping

This Clipping procedure is suitable for Concave Polygons. The process is almost same as Sutherland-Hodgeman algorithm except two minor changes. We process pair of vertices of polygon either clockwise or anticlockwise and test them against the Clip window boundary one by one. Let us assume that we are processing vertices in a clockwise direction.

1. If both vertices are outside then nothing is saved. (Because It means that edge is completely outside the clip window).
2. If both vertices are inside then we will save both. (Since it means that the edge connecting those vertices is completely inside the clip window).
3. If the first vertex is inside and the second vertex is outside then the first vertex is saved as it is and the intersection point of that polygon edge with clip window is saved. After doing so *follow Window boundary in a clockwise direction.*
4. If the second vertex is inside and the first vertex is outside then we will save the second vertex. We will also save the intersection point of the polygon edge with a clip window. After doing so, *follow Polygon boundary*.

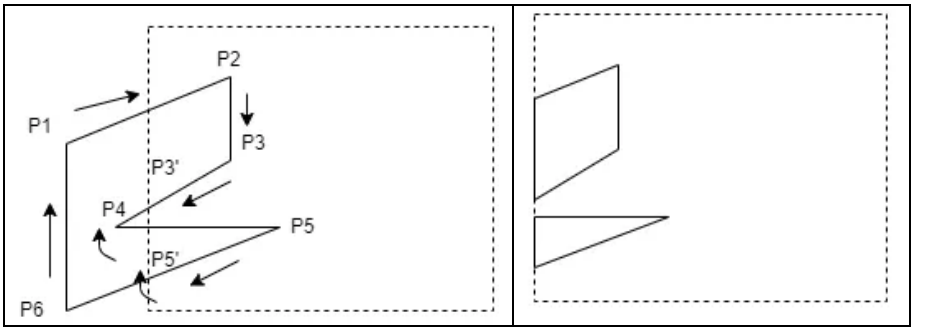
Example: Consider the following polygon P1, P2, P3, P4, P5, P6 with edges P1P2, P2P3, P3P4, P4P5, P5P6 and P6P1. Clip it using Weiler-Atherton Polygon Clipping.



To apply this algorithm, we see if the edge is in-out or out-in. If the edge is out-in, then we follow side of polygon. If the edge is in-in, then we save intersecting point on window boundary.

1. P1P2 is out-in edge, following side of polygon.

2. P2P3 is in-in edge, we skip this and include this edge.
3. P3P4 is in-out edge, saving intersecting point on window boundary i.e. P3′.
4. P4P5 is out-in edge, following side of polygon.
5. P5P6 is in-out edge, saving intersecting point on window boundary i.e. P5′.
6. P6P1 is out-out edge, we skip this and exclude this edge.



Here two separate polygon areas generated without any extra line connecting two polygons as final output shown in the above figure.

EXERCISE

1. What is Clipping? What is the need of clipping? Explain different types of clipping briefly.
2. How World coordinates are transformed to Viewport Coordinates? Derive the equation for the same.
3. Explain the following terms:
   a. Window b. Viewport
   c. Zooming d. Panning

5. 3D TRANFORMATIONS & VIEWING

3D Transformations take place in a three dimensional plane.3D Transformations are important and a bit more complex than 2D Transformations. The following sections discusses the all types of transformations in 3D context.

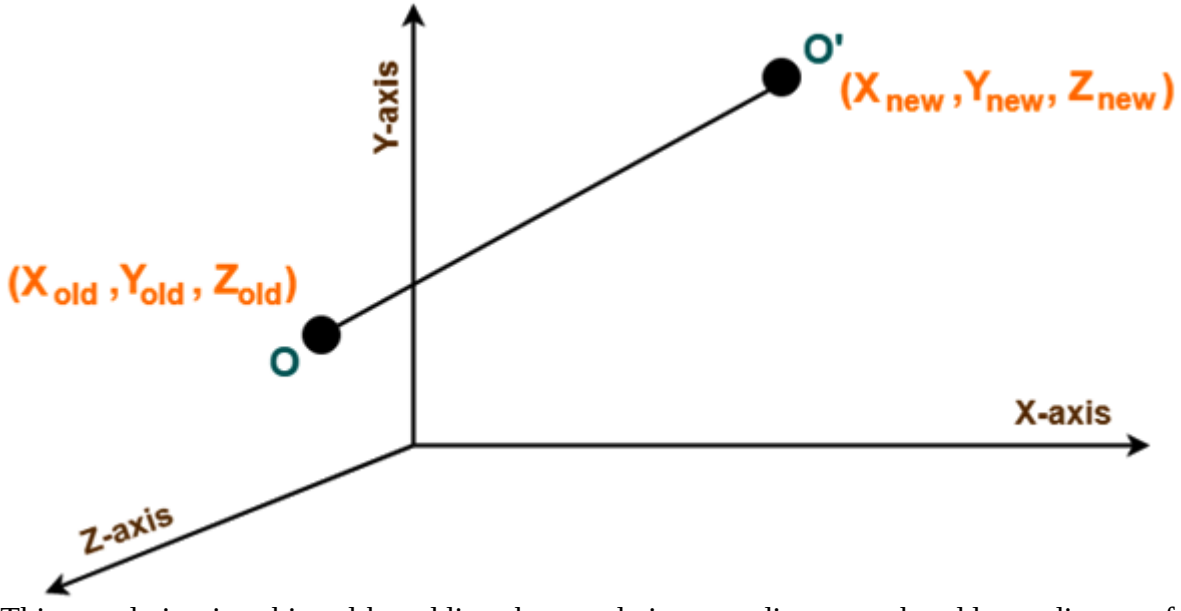
1. TRANSLATION

Translation, as defined in 2D, it is moving the object by some distance. In 3D translation, there are three Translation vectors.
Let the,

- Initial coordinates of the object O = (X_{old}, Y_{old}, Z_{old})
- New coordinates of the object O after translation = (X_{new}, Y_{new}, Z_{old})
- Translation vector or Shift vector = (T_x, T_y, T_z)

Given a Translation vector (T_x, T_y, T_z)

- T_x defines the distance the X_{old} coordinate has to be moved.
- T_y defines the distance the Y_{old} coordinate has to be moved.
- T_z defines the distance the Z_{old} coordinate has to be moved.



This translation is achieved by adding the translation coordinates to the old coordinates of the object as:

- $X_{new} = X_{old} + T_x$ (This denotes translation towards X axis)
- $Y_{new} = Y_{old} + T_y$ (This denotes translation towards Y axis)
- $Z_{new} = Z_{old} + T_z$ (This denotes translation towards Z axis)

In Matrix form, the above translation equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

Example: Given a 3D object with coordinate points A(0, 3, 1), B(3, 3, 2), C(3, 0, 0), D(0, 0, 0). Apply the translation with the distance 1 towards X axis, 1 towards Y axis and 2 towards Z axis and obtain the new coordinates of the object.

Solution:

- Old coordinates of the object = A (0, 3, 1), B(3, 3, 2), C(3, 0, 0), D(0, 0, 0)
- Translation vector = $(T_x, T_y, T_z) = (1, 1, 2)$

For Coordinates A(0, 3, 1)

Applying the translation equations, we have-

- $X_{new} = X_{old} + T_x = 0 + 1 = 1$
- $Y_{new} = Y_{old} + T_y = 3 + 1 = 4$
- $Z_{new} = Z_{old} + T_z = 1 + 2 = 3$

Thus, New coordinates of A = (1, 4, 3).

For Coordinates B(3, 3, 2)

Applying the translation equations, we have-

- $X_{new} = X_{old} + T_x = 3 + 1 = 4$
- $Y_{new} = Y_{old} + T_y = 3 + 1 = 4$
- $Z_{new} = Z_{old} + T_z = 2 + 2 = 4$

Thus, New coordinates of B = (4, 4, 4).

For Coordinates C(3, 0, 0)

Applying the translation equations, we have-

- $X_{new} = X_{old} + T_x = 3 + 1 = 4$
- $Y_{new} = Y_{old} + T_y = 0 + 1 = 1$
- $Z_{new} = Z_{old} + T_z = 0 + 2 = 2$

Thus, New coordinates of C = (4, 1, 2).

For Coordinates D(0, 0, 0)

Applying the translation equations, we have-

- $X_{new} = X_{old} + T_x = 0 + 1 = 1$

- $Y_{new} = Y_{old} + T_y = 0 + 1 = 1$
- $Z_{new} = Z_{old} + T_z = 0 + 2 = 2$

Thus, New coordinates of D = (1, 1, 2).

Thus, New coordinates of the object = A (1, 4, 3), B(4, 4, 4), C(4, 1, 2), D(1, 1, 2).

2. ROTATION

Consider a point object O has to be rotated from one angle to another in a 3D plane.

- Initial coordinates of the object O = (X_{old}, Y_{old}, Z_{old})
- Initial angle of the object O with respect to origin = Φ
- Rotation angle = θ
- New coordinates of the object O after rotation = (X_{new}, Y_{new}, Z_{new})

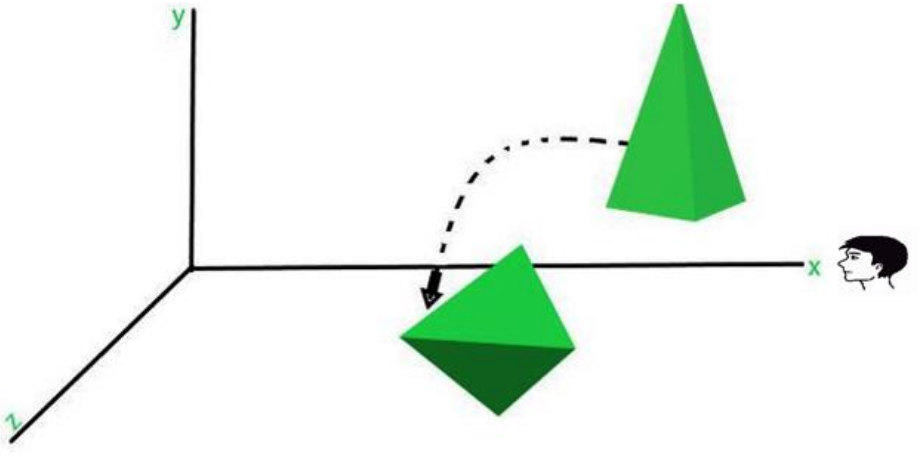
In 3 dimensions, there are 3 possible types of rotation as under:

- X-axis Rotation
- Y-axis Rotation
- Z-axis Rotation

For X-Axis Rotation

This rotation is achieved by using the following rotation equations where X coordinates remain unchanged:

- $X_{new} = X_{old}$
- $Y_{new} = Y_{old} \times \cos\theta - Z_{old} \times \sin\theta$
- $Z_{new} = Y_{old} \times \sin\theta + Z_{old} \times \cos\theta$



Rotation about X axis

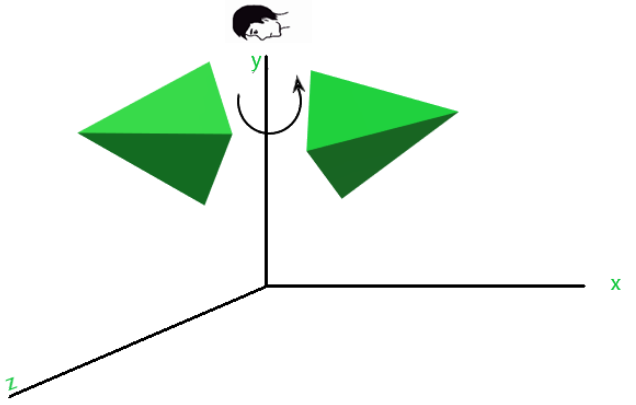
In Matrix form, the above rotation equations may be represented as,

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

For Y-Axis Rotation

This rotation is achieved by using the following rotation equations where Y coordinates remain unchanged:

- $X_{new} = Z_{old}$ x sinθ + X_{old} x cosθ
- $Y_{new} = Y_{old}$
- $Z_{new} = Y_{old}$ x cosθ – X_{old} x sinθ



Rotation about Y axis

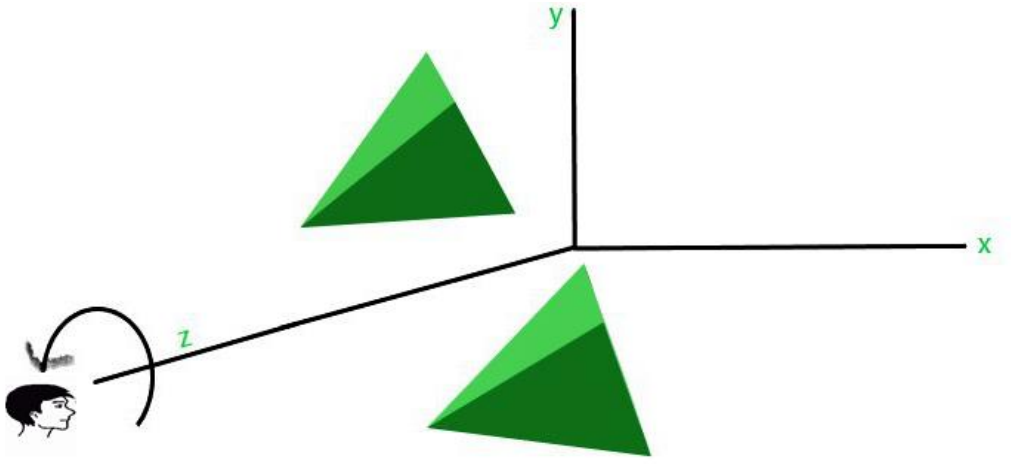
In Matrix form, the above rotation equations may be represented as,

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

For Z-Axis Rotation

This rotation is achieved by using the following rotation equations where Z coordinates remain unchanged:

- $X_{new} = X_{old}$ x cosθ – Y_{old} x sinθ
- $Y_{new} = X_{old}$ x sinθ + Y_{old} x cosθ
- $Z_{new} = Z_{old}$

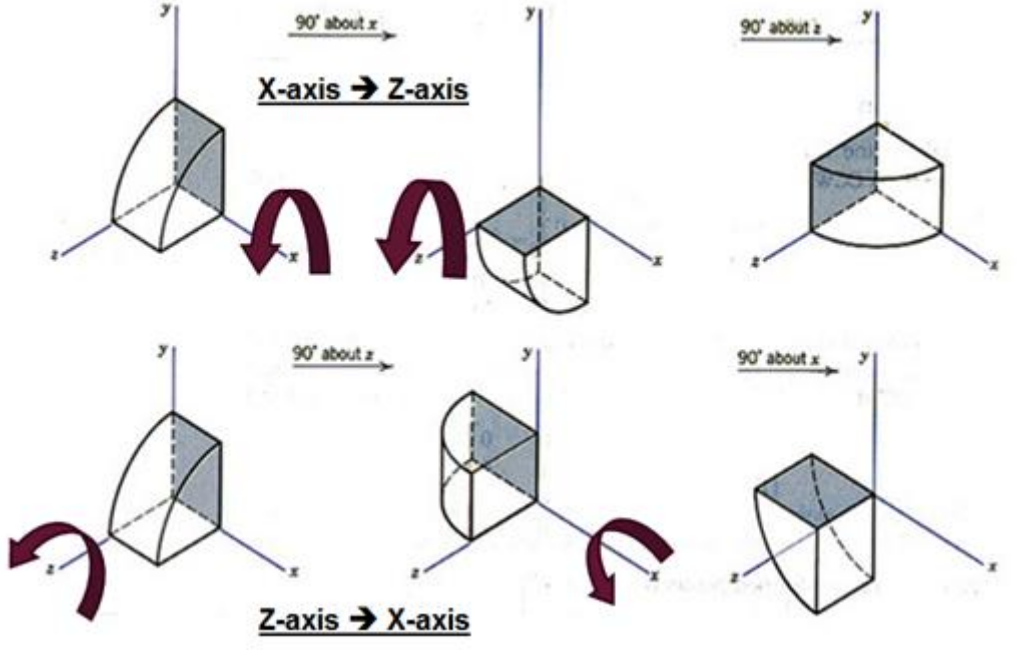


Rotation about Z axis

In Matrix form, the above rotation equations may be represented as,

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

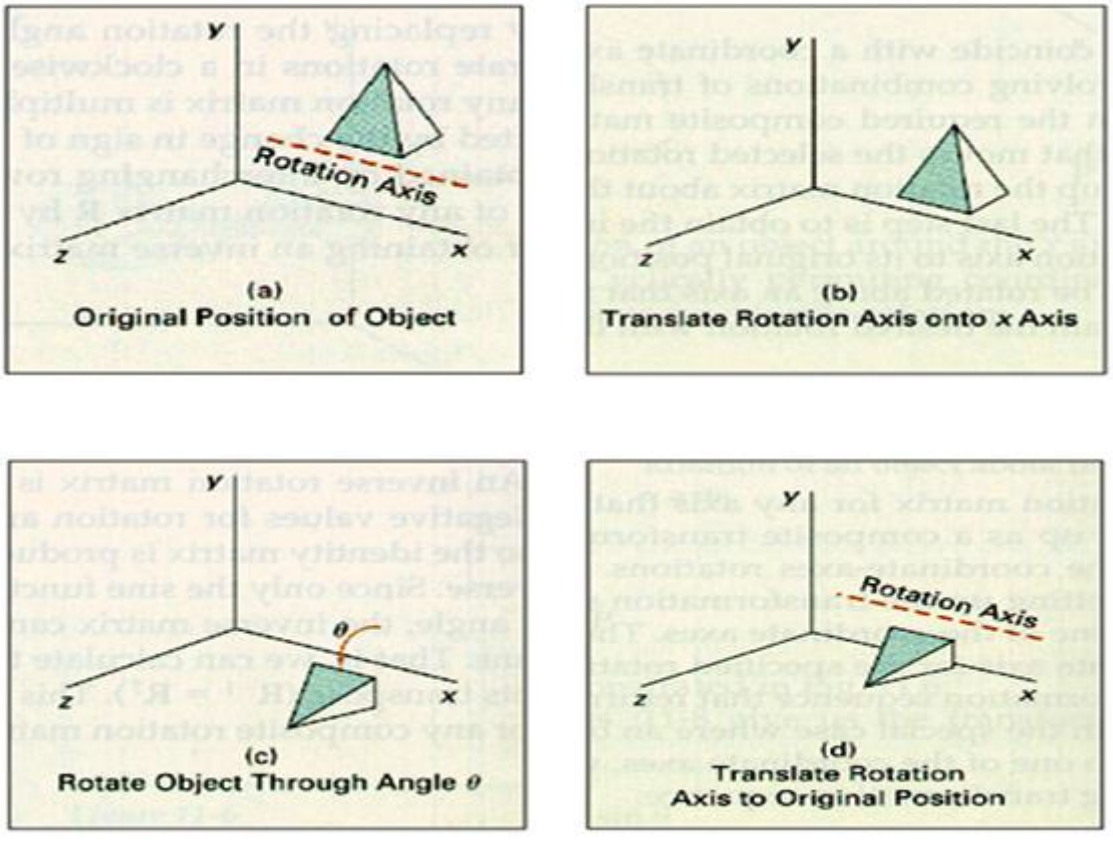
Here, the order of rotation affects the final position of the object. This can be seen in the following figure:



Apart from this rotation with reference to axis, there are other two types of rotations for 3D objects: one is rotation about an axis which is parallel to any one of the coordinate axis and the other one is rotation about any arbitrary axis which is neither parallel to any of the axis nor passing through origin.

2.1 Rotation about an Axis that is Parallel to One of the Coordinate Axis

1. Translate the object so that the rotation axis coincides with the parallel coordinate axis.
2. Perform the specified rotation about that coinciding axis.
3. Translate the object so that the rotation axis is moved back to its original position.



2.2 Rotation about an Arbitrary Axis

To rotate an object with reference to any arbitrary axis, we will have to coincide that arbitrary axis with z axis. For that, follow the steps given below:

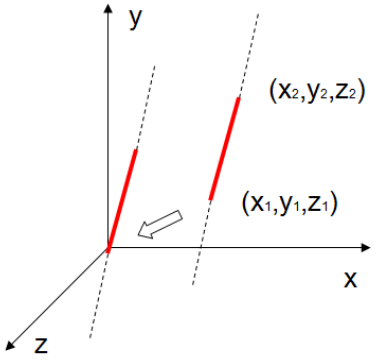
1. Translate (x1, y1, z1) to the origin.
2. Rotate object with an angle of α around X axis.
3. Rotate object with an angle of β around Y axis.
4. Rotate object with an angle of θ around Z axis.

5. Rotate object with an angle of -β around Y axis.
6. Rotate object with an angle of -α around X axis.
7. Translate the rotation axis to the original position.

The transformation equation will be as follows:

$$R(\theta) = T^{-1}R_x(-\alpha)R_y(-\beta)R_z(\theta)R_y(\beta)R_x(\alpha)T$$

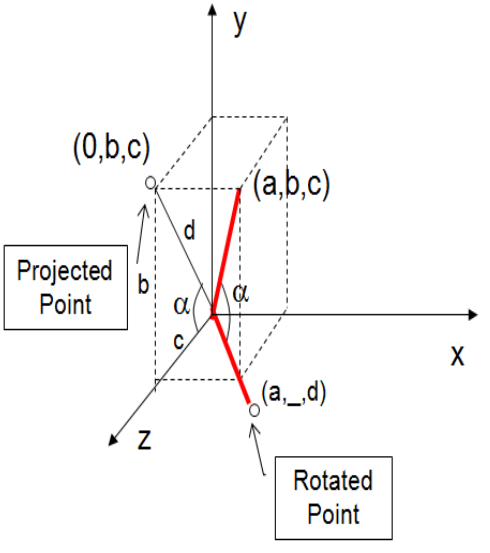
Step-1: Translate axis end points (x1, y1, z1) to the origin. Let's take the coordinates after translation as (0,0,0) and (a,b,c).



$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step-1

Step-2: Rotate the line with an angle of α about X axis, until it lies on the *XZ* plane. To do this, get the angle α by projecting the line on YZ plane as shown in figure. By projection, (a,b,c) will become (0,b,c) on YZ plane. α is the angle made by projected line with Z-axis on YZ plane. Rotating the line with an angle of α will give us (a, *some value*, d). Taking the projection of the line after rotation onto XZ plane will give us (a, 0, d).



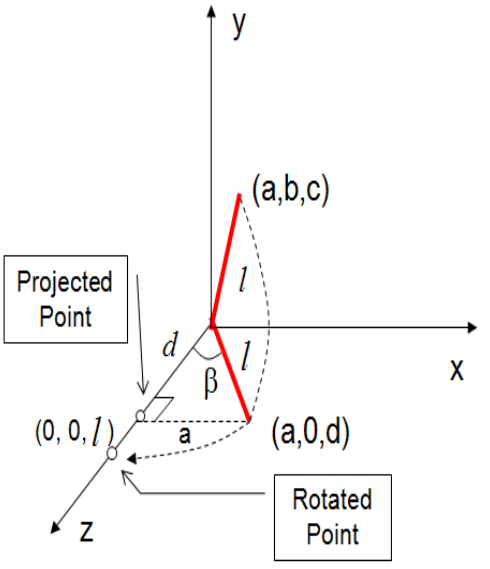
$$\sin\alpha = \frac{b}{\sqrt{b^2+c^2}} = \frac{b}{d}$$

$$\cos\alpha = \frac{c}{\sqrt{b^2+c^2}} = \frac{c}{d}$$

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step-2

Step-3: Rotate the projected line with an angle of β about Y axis so that it coincides with Z axis. β is the angle made by the projected line with Z-axis on YZ plane. Here, we will have coordinates (0, 0, *l*) after β rotation.



$$\sin\beta = \frac{a}{l}, \quad \cos\beta = \frac{d}{l}$$

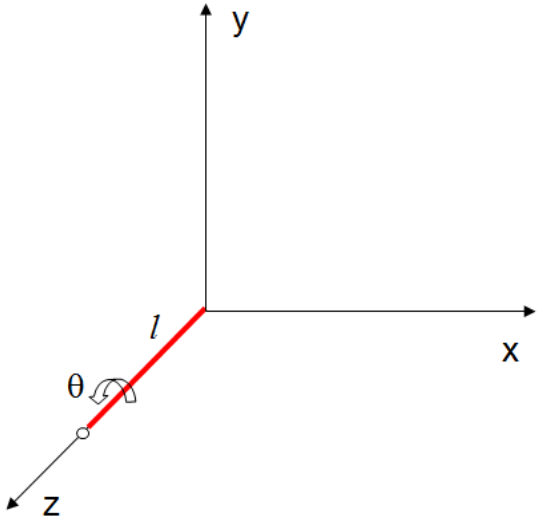
$$l^2 = a^2 + b^2 + c^2 = a^2 + d^2$$

$$d = \sqrt{b^2 + c^2}$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d/l & 0 & -a/l & 0 \\ 0 & 1 & 0 & 0 \\ a/l & 0 & d/l & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step-3

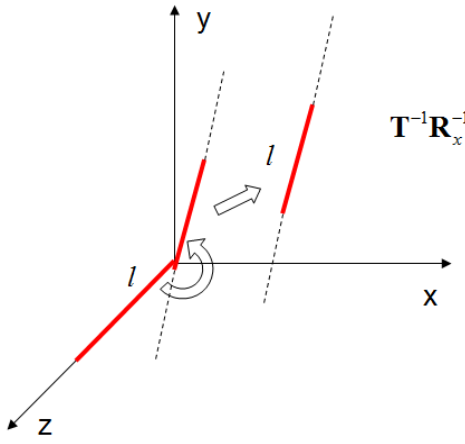
Step-4: Rotate line with an angle of θ around Z axis



$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step-4

Step-5, 6, 7: Apply the reverse transformation to place the axis back in its original position



$$\mathbf{T}^{-1}\mathbf{R}_x^{-1}(\alpha)\mathbf{R}_y^{-1}(\beta)=\begin{bmatrix}1&0&0&-x_1\\0&1&0&-y_1\\0&0&1&-z_1\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&\cos\alpha&\sin\alpha&0\\0&-\sin\alpha&\cos\alpha&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}\cos\beta&0&\sin\beta&0\\0&1&0&0\\-\sin\beta&0&\cos\beta&0\\0&0&0&1\end{bmatrix}$$

Step-5, 6, 7

<u>Example 1: Given a homogeneous point (1, 2, 3). Apply rotation 90 degree towards X, Y and Z axis and find out the new coordinate points.</u>

Solution:

Given that,

- Old coordinates = (X_{old}, Y_{old}, Z_{old}) = (1, 2, 3)
- Rotation angle = θ = 90º

For X-Axis Rotation:

Let the new coordinates after rotation = (X_{new}, Y_{new}, Z_{new}).

Applying the rotation equations, we have-

- $X_{new} = X_{old} = 1$
- $Y_{new} = Y_{old} \cos\theta - Z_{old} \sin\theta = 2 \cos90° - 3 \sin90° = 2 \times 0 - 3 \times 1 = -3$
- $Z_{new} = Y_{old} \sin\theta + Z_{old} \cos\theta = 2 \sin90° + 3 \cos90° = 2 \times 1 + 3 \times 0 = 2$

Thus, New coordinates after rotation = (1, -3, 2).

For Y-Axis Rotation:

Let the new coordinates after rotation = (X_{new}, Y_{new}, Z_{new}).

Applying the rotation equations, we have-

- $X_{new} = Z_{old} \sin\theta + X_{old} \cos\theta = 3 \sin90° + 1 \cos90° = 3 \times 1 + 1 \times 0 = 3$
- $Y_{new} = Y_{old} = 2$
- $Z_{new} = Y_{old} \cos\theta - X_{old} \sin\theta = 2 \cos90° - 1 \sin90° = 2 \times 0 - 1 \times 1 = -1$

Thus, New coordinates after rotation = (3, 2, -1).

For Z-Axis Rotation:

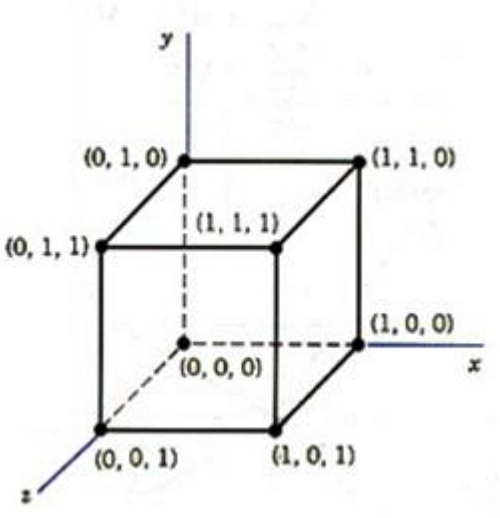
Let the new coordinates after rotation = (X_{new}, Y_{new}, Z_{new}).

Applying the rotation equations, we have-

- $X_{new} = X_{old} \cos\theta - Y_{old} \sin\theta = 1 \cos 90° - 2 \sin 90° = 1 \times 0 - 2 \times 1 = -2$
- $Y_{new} = X_{old} \sin\theta + Y_{old} \cos\theta = 1 \sin 90° + 2 \cos 90° = 1 \times 1 + 2 \times 0 = 1$
- $Z_{new} = Z_{old} = 3$

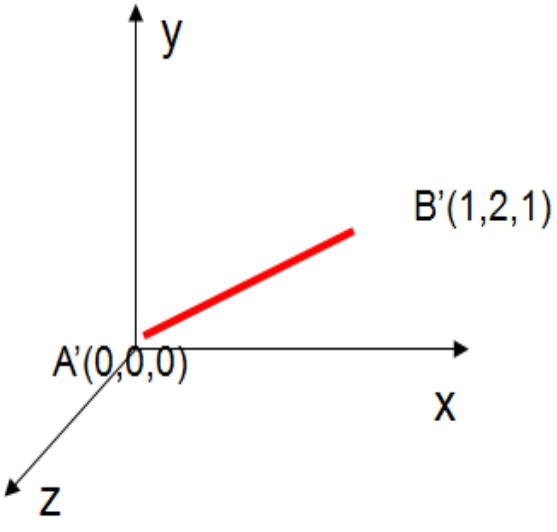
Thus, New coordinates after rotation = (-2, 1, 3).

Example 2: Find the new coordinates of a unit cube 90º rotated about an axis defined by its endpoints A(2,1,0) and B(3,3,1).



Solution:

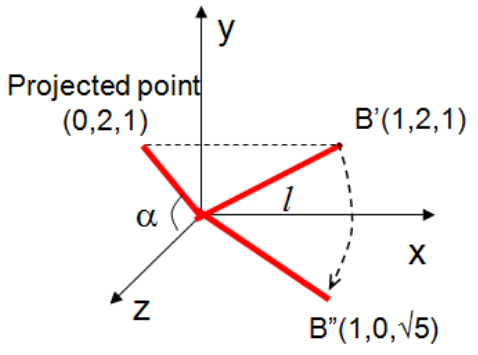
Step-1: Translate point A to the origin



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

Step-1

Step-2: Rotate axis *A'B'* about the x-axis by and angle α, until it lies on the *xz* plane.



$$\sin\alpha = \frac{2}{\sqrt{2^2+1^2}} = \frac{2}{\sqrt{5}} = \frac{2\sqrt{5}}{5}$$

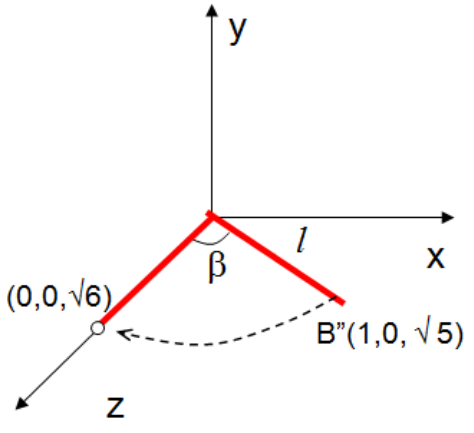
$$\cos\alpha = \frac{1}{\sqrt{5}} = \frac{\sqrt{5}}{5}$$

$$l = \sqrt{1^2+2^2+1^2} = \sqrt{6}$$

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{5}}{5} & -\frac{2\sqrt{5}}{5} & 0 \\ 0 & \frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step-2

Step-3: Rotate axis *A'B''* about the y-axis by and angle β, until it coincides with the z-axis.



$$\sin\beta = \frac{1}{\sqrt{6}} = \frac{\sqrt{6}}{6}$$

$$\cos\beta = \frac{\sqrt{5}}{\sqrt{6}} = \frac{\sqrt{30}}{6}$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \frac{\sqrt{30}}{6} & 0 & -\frac{\sqrt{6}}{6} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{\sqrt{6}}{6} & 0 & \frac{\sqrt{30}}{6} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step-3

Step 4: Rotate the cube 90° about the z-axis

$$\mathbf{R}_z(90°) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Finally, the concatenated rotation matrix about the arbitrary axis AB becomes,

$$\mathbf{R(\theta) = T^{-1}R_x(-\alpha)R_y(-\beta)R_z(90)R_y(\beta)R_x(\alpha)T}$$

$$\mathbf{R}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{5}}{5} & \frac{2\sqrt{5}}{5} & 0 \\ 0 & -\frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{30}}{6} & 0 & \frac{\sqrt{6}}{6} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\sqrt{6}}{6} & 0 & \frac{\sqrt{30}}{6} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{\sqrt{30}}{6} & 0 & -\frac{\sqrt{6}}{6} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{\sqrt{6}}{6} & 0 & \frac{\sqrt{30}}{6} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{5}}{5} & -\frac{2\sqrt{5}}{5} & 0 \\ 0 & \frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.166 & -0.075 & 0.983 & 1.742 \\ 0.742 & 0.667 & 0.075 & -1.151 \\ -0.650 & 0.741 & 0.167 & 0.560 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying $\mathbf{R}(\theta)$ by the point matrix of the original cube as P'= $\mathbf{R}(\theta)$*[P]

$$[\mathbf{P'}] = \begin{bmatrix} 0.166 & -0.075 & 0.983 & 1.742 \\ 0.742 & 0.667 & 0.075 & -1.151 \\ -0.650 & 0.741 & 0.167 & 0.560 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2.650 & 1.667 & 1.834 & 2.816 & 2.725 & 1.742 & 1.909 & 2.891 \\ -0.558 & -0.484 & 0.258 & 0.184 & -1.225 & -1.151 & -0.409 & -0.483 \\ 1.467 & 1.301 & 0.650 & 0.817 & 0.726 & 0.560 & -0.091 & 0.076 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

3. SCALING

- Scaling may be used to increase or reduce the size of object.
- Scaling subjects the coordinate points of the original object to change.
- Scaling factor determines whether the object size is to be increased or reduced.
- If scaling factor > 1, then the object size is increased.
- If scaling factor < 1, then the object size is reduced.

Consider a point object O has to be scaled in a 3D plane. Let the,

- Initial coordinates of the object O = (X_{old}, Y_{old},Z_{old})
- Scaling factor for X-axis = S_x
- Scaling factor for Y-axis = S_y
- Scaling factor for Z-axis = S_z
- New coordinates of the object O after scaling = (X_{new}, Y_{new}, Z_{new})

This scaling is achieved by using the following scaling equations:

- $X_{new} = X_{old} \times S_x$
- $Y_{new} = Y_{old} \times S_y$
- $Z_{new} = Z_{old} \times S_z$

In Matrix form, the above scaling equations may be represented as,

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

Example: Given a 3D object with coordinate points A(0, 3, 3), B(3, 3, 6), C(3, 0, 1), D(0, 0, 0). Apply the scaling parameter 2 towards X axis, 3 towards Y axis and 3 towards Z axis and obtain the new coordinates of the object.

Solution:

Given that,

- Old coordinates of the object = A (0, 3, 3), B(3, 3, 6), C(3, 0, 1), D(0, 0, 0)
- Scaling factor along X axis = 2
- Scaling factor along Y axis = 3
- Scaling factor along Z axis = 3

For Coordinates A(0, 3, 3)

Let the new coordinates of A after scaling = (X_{new}, Y_{new}, Z_{new}).

Applying the scaling equations, we have-

- $X_{new} = X_{old} \times S_x = 0 \times 2 = 0$
- $Y_{new} = Y_{old} \times S_y = 3 \times 3 = 9$
- $Z_{new} = Z_{old} \times S_z = 3 \times 3 = 9$

Thus, New coordinates of corner A after scaling = (0, 9, 9).

For Coordinates B(3, 3, 6)

Let the new coordinates of B after scaling = (X_{new}, Y_{new}, Z_{new}).

Applying the scaling equations, we have-

- $X_{new} = X_{old} \times S_x = 3 \times 2 = 6$
- $Y_{new} = Y_{old} \times S_y = 3 \times 3 = 9$
- $Z_{new} = Z_{old} \times S_z = 6 \times 3 = 18$

Thus, New coordinates of corner B after scaling = (6, 9, 18).

For Coordinates C(3, 0, 1)

Let the new coordinates of C after scaling = (X_{new}, Y_{new}, Z_{new}).

Applying the scaling equations, we have-

- $X_{new} = X_{old} \times S_x = 3 \times 2 = 6$
- $Y_{new} = Y_{old} \times S_y = 0 \times 3 = 0$
- $Z_{new} = Z_{old} \times S_z = 1 \times 3 = 3$

Thus, New coordinates of corner C after scaling = (6, 0, 3).

For Coordinates D(0, 0, 0)

Let the new coordinates of D after scaling = (X_{new}, Y_{new}, Z_{new}).

Applying the scaling equations, we have-

- $X_{new} = X_{old} \times S_x = 0 \times 2 = 0$
- $Y_{new} = Y_{old} \times S_y = 0 \times 3 = 0$
- $Z_{new} = Z_{old} \times S_z = 0 \times 3 = 0$

Thus, New coordinates of corner D after scaling = (0, 0, 0).

4. REFLECTION

- Reflection is a kind of rotation where the angle of rotation is 180 degree.
- The reflected object is always formed on the other side of mirror.
- The size of reflected object is same as the size of original object.

Consider a point object O has to be reflected in a 3D plane. Let the,

- Initial coordinates of the object O = (X_{old}, Y_{old}, Z_{old})
- New coordinates of the reflected object O after reflection = (X_{new}, Y_{new},Z_{new})

In 3 dimensions, there are 3 possible types of reflection:

1. Reflection relative to XY plane
2. Reflection relative to YZ plane
3. Reflection relative to XZ plane

In reflection with reference to any of the plane, the third remaining coordinate becomes negative. The following sections discuss all three types of reflections.

4.1 Reflection Relative to XY Plane

This reflection is achieved by using the following reflection equations:

- $X_{new} = X_{old}$
- $Y_{new} = Y_{old}$
- $Z_{new} = -Z_{old}$

In Matrix form, the above reflection equations may be represented as,

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

Reflection with respect to XY plane

4.2 Reflection Relative to YZ Plane

This reflection is achieved by using the following reflection equations:

- $X_{new} = -X_{old}$
- $Y_{new} = Y_{old}$
- $Z_{new} = Z_{old}$

In Matrix form, the above reflection equations may be represented as,

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

Reflection with respect to YZ plane

4.3 Reflection Relative to XZ Plane

This reflection is achieved by using the following reflection equation:

- $X_{new} = X_{old}$
- $Y_{new} = -Y_{old}$
- $Z_{new} = Z_{old}$

In Matrix form, the above reflection equations may be represented as,

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

Reflection with respect to YZ plane

Example: Given a 3D triangle with coordinate points A(3, 4, 1), B(6, 4, 2), C(5, 6, 3). Apply the reflection on the XY plane and find out the new coordinates of the object.

Solution:

Given that,

- Old corner coordinates of the triangle = A (3, 4, 1), B(6, 4, 2), C(5, 6, 3)
- Reflection has to be taken on the XY plane

For Coordinates A(3, 4, 1)

Let the new coordinates of corner A after reflection = (X_{new}, Y_{new}, Z_{new}).

Applying the reflection equations, we have

- $X_{new} = X_{old} = 3$
- $Y_{new} = Y_{old} = 4$
- $Z_{new} = -Z_{old} = -1$

Thus, New coordinates of corner A after reflection = (3, 4, -1).

For Coordinates B(6, 4, 2)

Let the new coordinates of corner B after reflection = (X_{new}, Y_{new}, Z_{new}).

Applying the reflection equations, we have

- $X_{new} = X_{old} = 6$
- $Y_{new} = Y_{old} = 4$
- $Z_{new} = -Z_{old} = -2$

Thus, New coordinates of corner B after reflection = (6, 4, -2).

For Coordinates C(5, 6, 3)

Let the new coordinates of corner C after reflection = (X_{new}, Y_{new}, Z_{new}).

Applying the reflection equations, we have

- $X_{new} = X_{old} = 5$
- $Y_{new} = Y_{old} = 6$
- $Z_{new} = -Z_{old} = -3$

Thus, New coordinates of corner C after reflection = (5, 6, -3).

Thus, New coordinates of the triangle after reflection = A (3, 4, -1), B(6, 4, -2), C(5, 6, -3).

5. SHEARING

In a three dimensional plane, the object size can be changed along X direction, Y direction as well as Z direction. So, there are three types of shearing:

1. Shearing in X direction
2. Shearing in Y direction
3. Shearing in Z direction

Consider a point object O has to be sheared in a 3D plane.

Let the,

- Initial coordinates of the object O = (X_{old}, Y_{old}, Z_{old})
- Shearing parameter towards X direction = Sh_x
- Shearing parameter towards Y direction = Sh_y
- Shearing parameter towards Z direction = Sh_z
- New coordinates of the object O after shearing = (X_{new}, Y_{new}, Z_{new})

5.1 Shearing in X Axis

Shearing in X axis is achieved by using the following shearing equations-

- $X_{new} = X_{old}$
- $Y_{new} = Y_{old} + Sh_y \times X_{old}$
- $Z_{new} = Z_{old} + Sh_z \times X_{old}$

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ Sh_y & 1 & 0 & 0 \\ Sh_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

Shearing in X axis

5.2 Shearing in Y Axis

Shearing in Y axis is achieved by using the following shearing equations-

- $X_{new} = X_{old} + Sh_x \times Y_{old}$
- $Y_{new} = Y_{old}$
- $Z_{new} = Z_{old} + Sh_z \times Y_{old}$

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Sh_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & Sh_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

Shearing in Y axis

5.3 Shearing in Z Axis

Shearing in Z axis is achieved by using the following shearing equations-

- $X_{new} = X_{old} + Sh_x \times Z_{old}$
- $Y_{new} = Y_{old} + Sh_y \times Z_{old}$
- $Z_{new} = Z_{old}$

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & Sh_x & 0 \\ 0 & 1 & Sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

Shearing in Z axis

Example: Given a 3D triangle with points (0, 0, 0), (1, 1, 2) and (1, 1, 3). Apply shear parameter 2 on X axis, 2 on Y axis and 3 on Z axis and find out the new coordinates of the object.

Solution:

Given that,

- Old corner coordinates of the triangle = A (0, 0, 0), B(1, 1, 2), C(1, 1, 3)
- Shearing parameter towards X direction (Sh_x) = 2
- Shearing parameter towards Y direction (Sh_y) = 2
- Shearing parameter towards Y direction (Sh_z) = 3

Shearing in X Axis

For Coordinates A(0, 0, 0)

Let the new coordinates of corner A after shearing = (X_{new}, Y_{new}, Z_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} = 0$
- $Y_{new} = Y_{old} + Sh_y \times X_{old} = 0 + 2 \times 0 = 0$
- $Z_{new} = Z_{old} + Sh_z \times X_{old} = 0 + 3 \times 0 = 0$

Thus, New coordinates of corner A after shearing = (0, 0, 0).

For Coordinates B(1, 1, 2)

Let the new coordinates of corner B after shearing = (X_{new}, Y_{new}, Z_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} = 1$
- $Y_{new} = Y_{old} + Sh_y \times X_{old} = 1 + 2 \times 1 = 3$
- $Z_{new} = Z_{old} + Sh_z \times X_{old} = 2 + 3 \times 1 = 5$

Thus, New coordinates of corner B after shearing = (1, 3, 5).

For Coordinates C(1, 1, 3)

Let the new coordinates of corner C after shearing = (X_{new}, Y_{new}, Z_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} = 1$
- $Y_{new} = Y_{old} + Sh_y \times X_{old} = 1 + 2 \times 1 = 3$
- $Z_{new} = Z_{old} + Sh_z \times X_{old} = 3 + 3 \times 1 = 6$

Thus, New coordinates of corner C after shearing = (1, 3, 6).

Thus, New coordinates of the triangle after shearing in X axis = A (0, 0, 0), B(1, 3, 5), C(1, 3, 6).

Shearing in Y Axis

For Coordinates A(0, 0, 0)

Let the new coordinates of corner A after shearing = (X_{new}, Y_{new}, Z_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Y_{old} = 0 + 2 \times 0 = 0$
- $Y_{new} = Y_{old} = 0$
- $Z_{new} = Z_{old} + Sh_z \times Y_{old} = 0 + 3 \times 0 = 0$

Thus, New coordinates of corner A after shearing = (0, 0, 0).

For Coordinates B(1, 1, 2)

Let the new coordinates of corner B after shearing = (X_{new}, Y_{new}, Z_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Y_{old} = 1 + 2 \times 1 = 3$
- $Y_{new} = Y_{old} = 1$
- $Z_{new} = Z_{old} + Sh_z \times Y_{old} = 2 + 3 \times 1 = 5$

Thus, New coordinates of corner B after shearing = (3, 1, 5).

For Coordinates C(1, 1, 3)

Let the new coordinates of corner C after shearing = (X_{new}, Y_{new}, Z_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Y_{old} = 1 + 2 \times 1 = 3$
- $Y_{new} = Y_{old} = 1$
- $Z_{new} = Z_{old} + Sh_z \times Y_{old} = 3 + 3 \times 1 = 6$

Thus, New coordinates of corner C after shearing = (3, 1, 6).

Thus, New coordinates of the triangle after shearing in Y axis = A (0, 0, 0), B(3, 1, 5), C(3, 1, 6).

Shearing in Z Axis

For Coordinates A(0, 0, 0)

Let the new coordinates of corner A after shearing = (X_{new}, Y_{new}, Z_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Z_{old} = 0 + 2 \times 0 = 0$
- $Y_{new} = Y_{old} + Sh_y \times Z_{old} = 0 + 2 \times 0 = 0$
- $Z_{new} = Z_{old} = 0$

Thus, New coordinates of corner A after shearing = (0, 0, 0).

For Coordinates B(1, 1, 2)

Let the new coordinates of corner B after shearing = (X_{new}, Y_{new}, Z_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Z_{old} = 1 + 2 \times 2 = 5$
- $Y_{new} = Y_{old} + Sh_y \times Z_{old} = 1 + 2 \times 2 = 5$
- $Z_{new} = Z_{old} = 2$

Thus, New coordinates of corner B after shearing = (5, 5, 2).

For Coordinates C(1, 1, 3)

Let the new coordinates of corner C after shearing = (X_{new}, Y_{new}, Z_{new}).

Applying the shearing equations, we have-

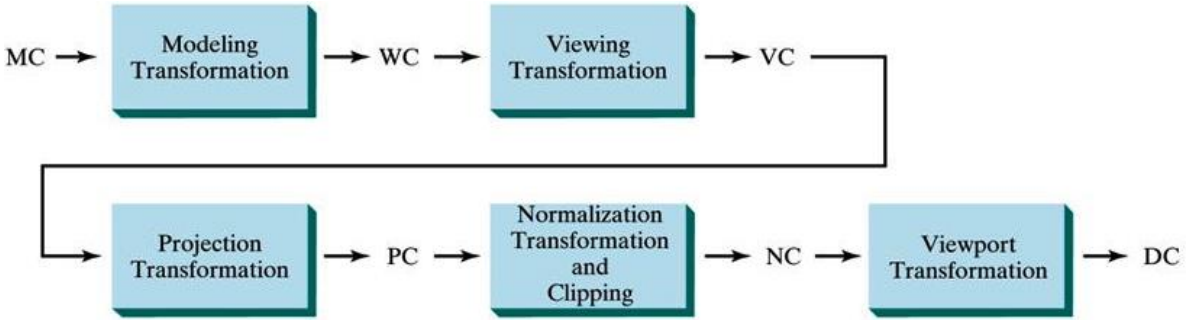
- $X_{new} = X_{old} + Sh_x \times Z_{old} = 1 + 2 \times 3 = 7$
- $Y_{new} = Y_{old} + Sh_y \times Z_{old} = 1 + 2 \times 3 = 7$
- $Z_{new} = Z_{old} = 3$

Thus, New coordinates of corner C after shearing = (7, 7, 3).

Thus, New coordinates of the triangle after shearing in Z axis = A (0, 0, 0), B(5, 5, 2), C(7, 7, 3).

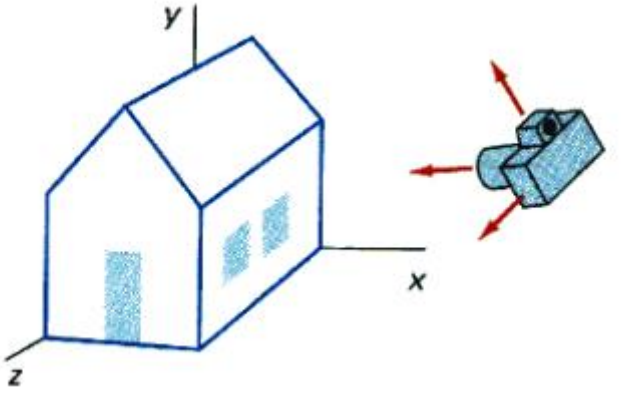
6. 3D VIEWING

In the 2D system, we use only two coordinates X and Y, but in 3D, an extra coordinate Z is added. 3D graphics techniques and their application are fundamental to the entertainment, games, and computer-aided design industries. It is a continuing area of research in scientific visualization. Furthermore, 3D graphics components are nowadays part of almost every personal computer and they are increasingly being used by other applications.

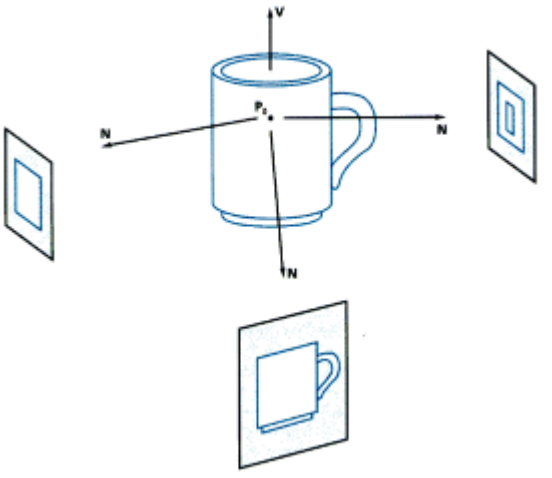


Viewing in 3D involves the following considerations:

- We can view an object from any spatial position. e.g. front of an object, back of the object, middle of a group of objects, inside an object etc.
- 3D descriptions of objects must be projected onto the flat viewing surface of the output device.
- The clipping boundaries enclose a volume of space.



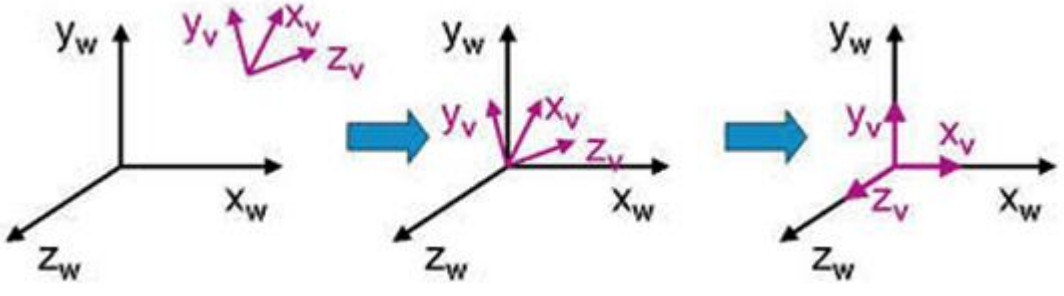
Modelling Transformation and Viewing Transformation can be done by 3D transformations we discussed in previous sections. The viewing-coordinate system is used in graphics packages as a reference for specifying the observer viewing position and the position of the projection plane to view the objects.



Projection operations convert the viewing-coordinate description (3D) to coordinate positions on the projection plane (2D) usually combined with clipping, visual-surface identification, and surface rendering. Workstation transformation maps the coordinate positions on the projection plane to the output device.

Viewing transformation conversion of object descriptions from world to viewing coordinates is equivalent to a transformation that superimposes the viewing reference frame onto the world frame using the basic geometric translate-rotate operations:

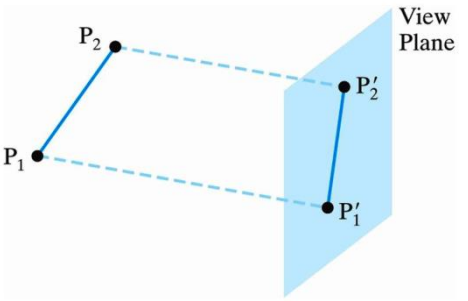
1. Translate the view reference point to the origin of the world-coordinate system.
2. Apply rotations to align the X_v, Y_v, and Z_v axes (viewing coordinate system) with the world X_w, Y_w, Z_w axes, respectively.



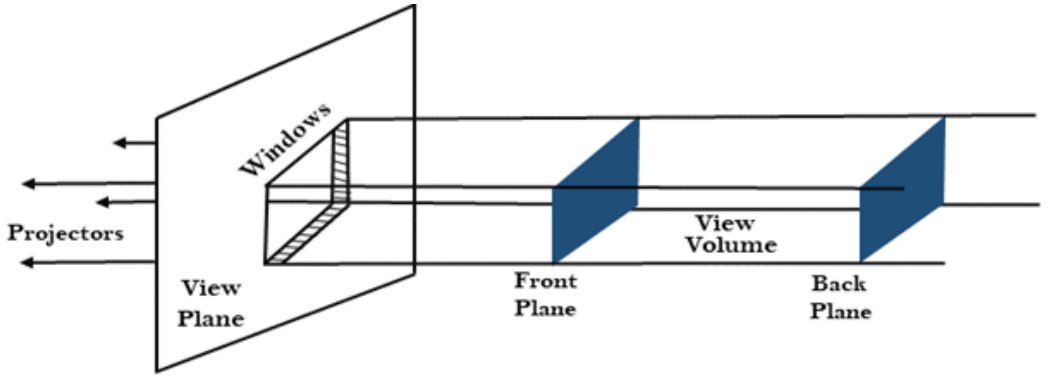
Projection operations convert the viewing-coordinate description of 3D to coordinate positions on the projection plane of 2D. There are 2 basic projection methods as discussed below:

6.1 Parallel Projections

It transforms object positions to the view plane along parallel lines. A parallel projection preserves relative proportions of objects. Accurate views of the various sides of an object are obtained with a parallel projection but they do not represent a realistic view.

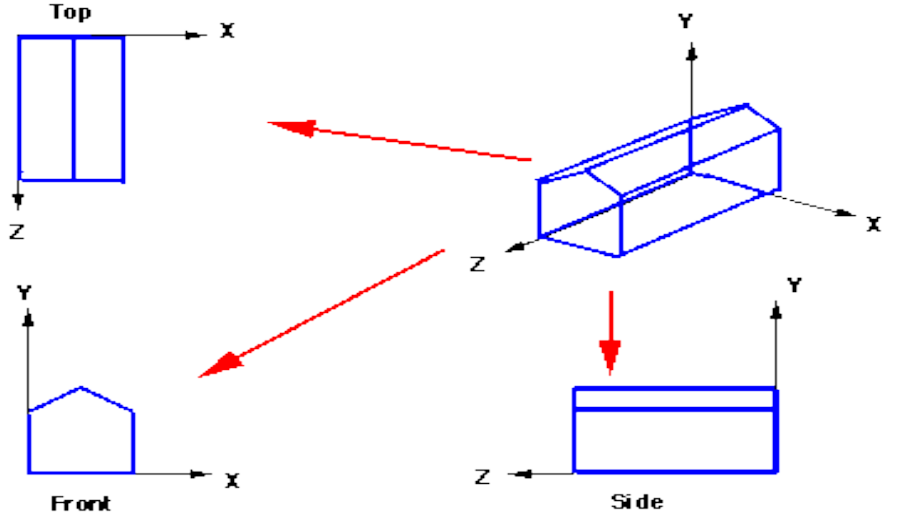


Parallel Projection are divided into Orthographic Parallel Projection and Oblique Projection categories:



Orthographic Parallel Projection

Orthographic parallel projections are carried out by projecting points along parallel lines that are perpendicular to the projection plane. However, oblique projections are obtained by projecting along parallel lines that are *NOT* perpendicular to the projection plane. Some special Orthographic Parallel Projections involve plan view (Top projection), side elevations, and isometric projections.

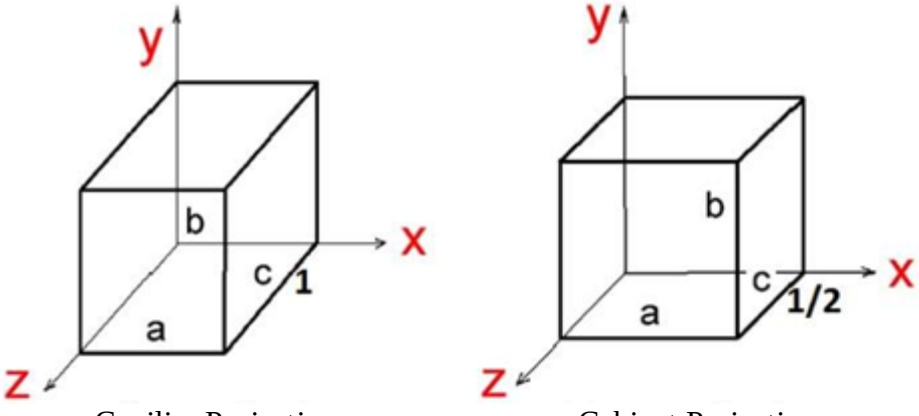


Top View (Plan), Front View (Elevation) and Side View of 3D object

In oblique projection, the direction of projection is not perpendicular to the projection plane. In oblique projection, we can view the object better than orthographic projection. There are two types of oblique projections − Cavalier and Cabinet.

The Cavalier projection makes an angle of 45° with the projection plane. The projection of a line perpendicular to the view plane has the same length as the original line in Cavalier projection.

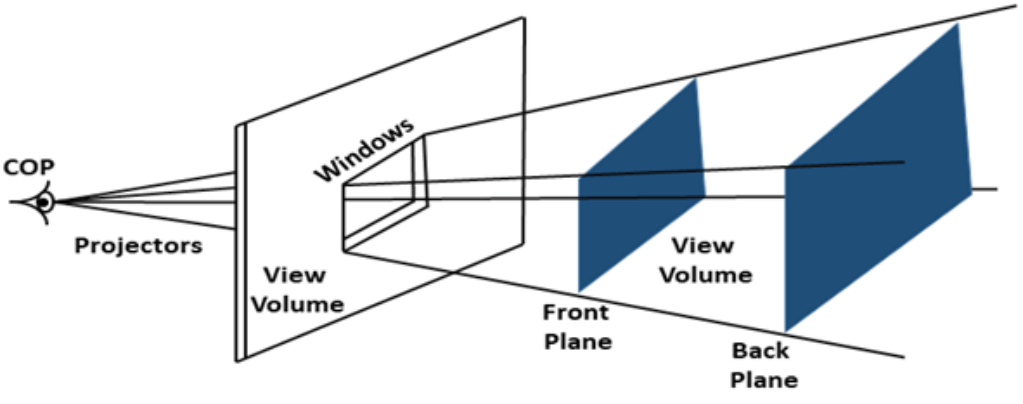
The Cabinet projection makes an angle of 63.4° with the projection plane. In Cabinet projection, lines perpendicular to the viewing surface are projected at 50% of their actual length. Both the projections are shown in the following figure:



Cavilier Projection Cabinet Projection

6.2 Perspective Projections

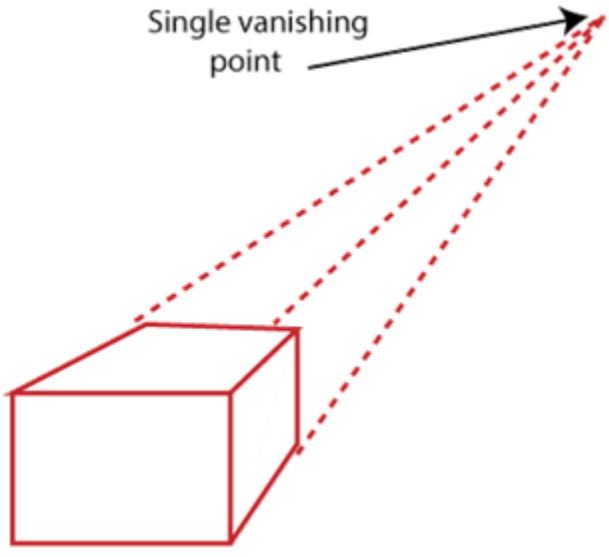
Perspective projection transforms object positions to the view plane while converging to a centre point of projection behind the projection plane. Perspective projection produces realistic views but does not preserve relative proportions. Projections of distant objects are smaller than the projections of objects of the same size that are closer to the projection plane.



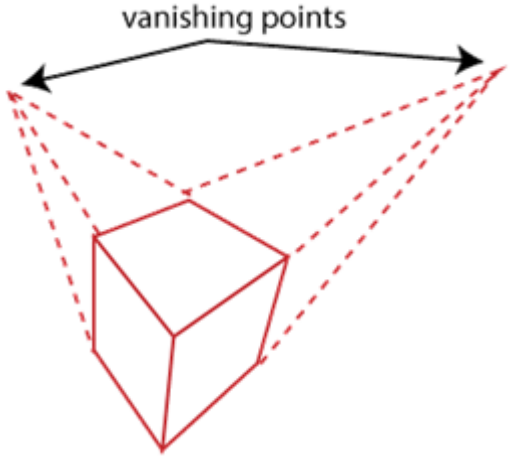
Perspective Projection

There are 3 types of perspective projections based on number of vanishing points. A Vanishing point is the point where projection lines converge and projection gets destroyed. Following figures explain the perspective projections based on the number of vanishing points.

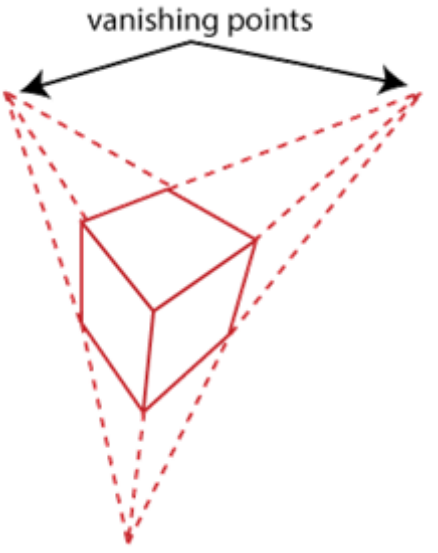
1. One point perspective projection is simple to draw.



2. Two point perspective projection gives better impression of depth.

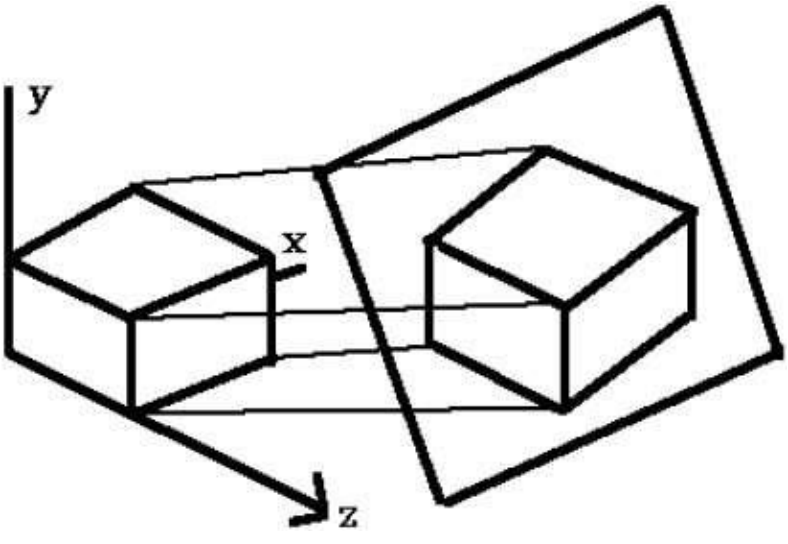


3. Three point perspective projection is most difficult to draw.



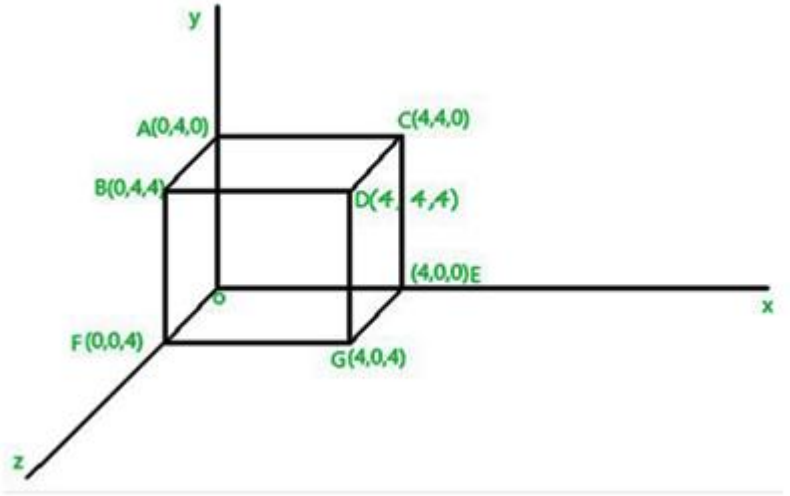
6.3 Isometric Projections

Orthographic projections that show more than one side of an object are called axonometric orthographic projections. The most common axonometric projection is an isometric projection where the projection plane intersects each coordinate axis in the model coordinate system at an equal distance. In this projection, parallelism of lines are preserved but angles are not preserved. The following figure shows isometric projection.



EXERCISE

1. Perform Rotation transformation over a cube 'OABCDEFG' given below and rotate it through 45* in the anticlockwise direction about the y-axis.



2. Rotate the above cube about an axis defined by its endpoints A(2,1,0) and B(3,3,1) by an angle of 45^0.
3. How does 3D Shearing differ from 2d Shearing? Explain with example.
4. Given a 3D triangle with coordinate points A(3, 4, 1), B(6, 4, 2), C(5, 6, 3). Apply the reflection on the XZ plane and find out the new coordinates of the object.

6. 3D OBJECT REPRESENTATIONS

3D object Representation schemes are little different than those of 2D images. They are divided into two categories as follows:

1. Boundary Representation (B-reps): It describes a three dimensional object as a set of surfaces that separate the object interior from the environment. For example, polygon facets and spline patches.
2. Space Partitioning representation: It describes the interior properties, by partitioning the spatial region containing an object into a set of small, non-overlapping, contiguous solids (usually cubes). For example, Octree Representation.

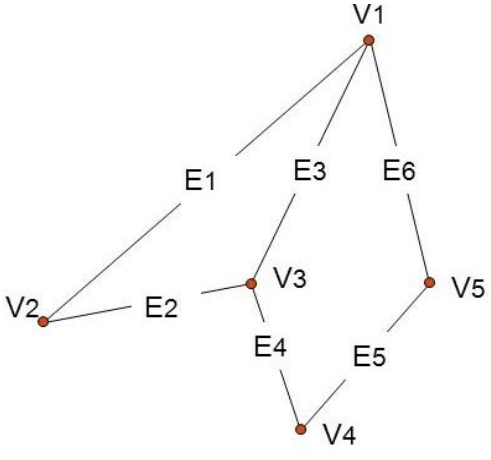
1. POLYGON SURFACES

Polygon surfaces, are boundary representations for a 3D graphics object, is a set of polygons that enclose the object interior. They are represented in the form of vector tables.

Polygon Tables: The polygon surface is specified with a set of vertex coordinates and associated attribute parameters. For each polygon input, the data are placed into tables, called polygon tables, that are to be used in the subsequent processing. Polygon data tables can be organized into two groups: Geometric tables and attribute tables.

The geometric tables contain vertex coordinates and parameters to identify the spatial orientation of the polygon surfaces whereas the attribute tables contain attribute information for an object such as parameters specifying the degree of transparency of the object, its surface reflectivity, texture characteristics etc. A convenient way of storing geometric data is to create three lists:

1. The Vertex Table: Coordinate values for each vertex in the object are stored in this table.
2. The Edge Table: It contains pointers back into the vertex table to identify the vertices for each polygon edge.
3. The Polygon Table: It contains pointers back into the edge table to identify the edges for each polygon. This is shown in following figure:



VERTEX TABLE	
V1	x1,y1,z1
V2	x2,y2,z2
V3	x3,y3,z3
V4	x4,y4,z4
V5	x5,y5,z5

EDGE TABLE	
E1	V1,V2
E2	V2,V3
E3	V3,V1
E4	V3,V4
E5	V4,V5
E6	V5,V1

POLYGON-SURFACE TABLE	
S1	E1,E2,E3
S2	E3,E4,E5,E6

Listing the geometric data in three tables provides a convenient reference to the individual components (vertices, edges and polygons) of each object. The object can be displayed efficiently by using data from the edge table to draw the component lines. Extra information can be added to the data tables for faster information extraction. For instance, edge table can be expanded to include forward points into the polygon table so that common edges between polygons can be identified more rapidly like given below.

E1 : V1, V2, S1

E2 : V2, V3, S1

E3 : V3, V1, S1, S2

E4 : V3, V4, S2

E5 : V4, V5, S2

E6 : V5, V1, S2

Above listing is useful for the rendering procedure that must vary surface shading smoothly across the edges from one polygon to the next. Similarly, the vertex table can be expanded so that vertices are cross-referenced to corresponding edges.

Additional geometric information that is stored in the data tables includes the slope for each edge and the coordinate extends for each polygon. Using vertices as input, we can calculate edge slopes, we can scan the coordinate values to identify the minimum and maximum x, y and z values for individual polygons.

The more information included in the data tables, more it will be easier to check for errors. Some of the tests that could be performed by a graphics package are:

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

2. PLANE EQUATIONS

To produce a display of a 3D object, we must process the input data representation for the object through several procedures such as, transformation of the modelling and world coordinate descriptions to viewing coordinates and then to device coordinates, identification of visible surfaces (also called removal of hidden surfaces) and finally the application of surface-rendering procedures.

For these processes, we need information about the spatial orientation of the individual surface components of the object. This information is obtained from the vertex coordinate value and the equations that describe the polygon planes. The equation for a plane surface is

$$Ax+By+Cz+D = 0 \text{ ----(1)}$$

Where (x, y, z) is any point on the plane, and the coefficients A, B, C and D are the constants describing the spatial properties of the plane. We can obtain the values of A, B, C and D by solving a set of three plane equations using the coordinate values for three non collinear points in the plane. For that, we can select three successive polygon vertices (x1, y1, z1), (x2, y2, z2) and (x3, y3, z3) and solve the following set of simultaneous linear plane equations for the ratios A/D, B/D and C/D.

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

The solution for this set of equations can be obtained in determinant form, using Cramer's rule as,

$$A = \begin{vmatrix} 1 & y1 & z1 \\ 1 & y2 & z2 \\ 1 & y3 & z3 \end{vmatrix} \quad B = \begin{vmatrix} x1 & 1 & z1 \\ x2 & 1 & z2 \\ x3 & 1 & z3 \end{vmatrix} \quad C = \begin{vmatrix} x1 & y1 & 1 \\ x2 & y2 & 1 \\ x3 & y3 & 1 \end{vmatrix} \quad D = \begin{vmatrix} x1 & y1 & z1 \\ x2 & y2 & z2 \\ x3 & y3 & z3 \end{vmatrix} ----(3)$$

Expanding the determinants, we can write the calculations for the plane coefficients in the following forms:

A = y1 (z2 –z3) + y2(z3 –z1) + y3 (z1 –z2)

B = z1 (x2 -x3) + z2 (x3 -x1) + z3 (x1 -x2)

C = x1 (y2 –y3) + x2 (y3 –y1) + x3 (y1 -y2)

D = -x1 (y2 z3 -y3 z2) - x2 (y3 z1 -y1 z3) - x3 (y1 z2 -y2 z1) ------(4)

As vertex values and other information are entered into the polygon data structure, values for A, B, C and D are computed for each polygon and stored with the other polygon data. Plane equations are used also to identify the position of spatial points relative to the plane surfaces of an object. For any given point (x, y, z) on a plane with parameters A, B, C, D, we have,

$$Ax+By+Cz+D \neq 0$$

We can identify the point as either inside or outside the plane surface according the sign (negative or positive) of Ax+By+Cz+D:

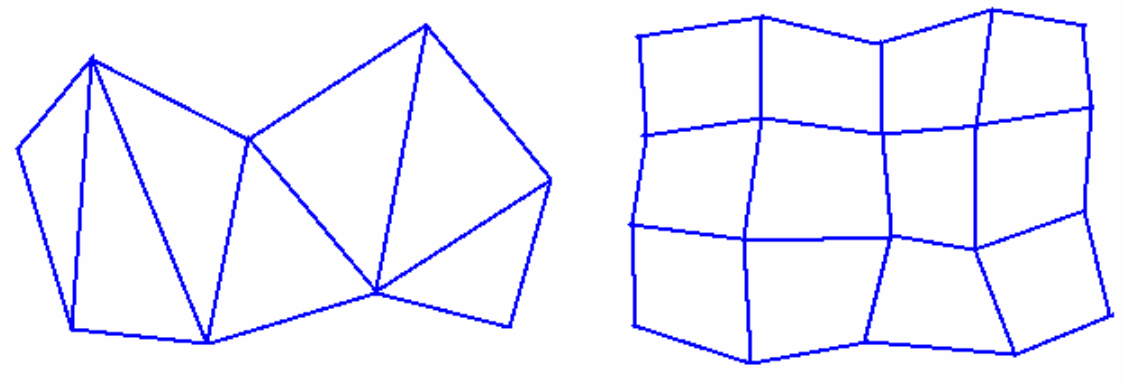
1. If Ax+By+Cz+D < 0, the point (x, y, z) is inside the surface.

2. If Ax+By+Cz+ D > 0, the point (x, y, z) is outside the surface.

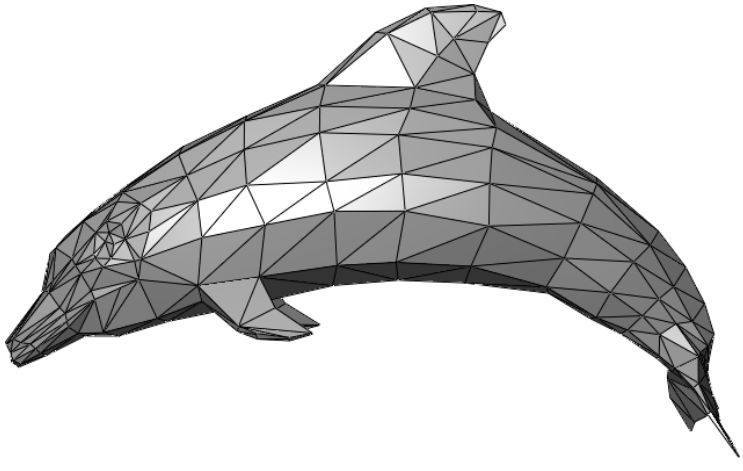
These inequality tests are valid in a right handed Cartesian system, provided the plane parameters A, B, C and D were calculated using vertices selected in a counter clockwise order when the viewing the surface is in an outside-to-inside direction.

3. POLYGON MESHES

A single plane surface can be specified with a function such as fill area but when object surfaces are to be tiled, it is more convenient to specify the surface facets with a mesh function. One type of polygon mesh function is the triangle strip. Here, in the below figure the mesh function produces n-2 connected triangles given the coordinates for n vertices. A triangle strip formed with 7 triangles connecting 9 vertices is shown below.



Another type of polygon mesh is a square strip which generates a mesh of (n-1) by (m-1) quadrilaterals, given the coordinates for an n by m array of vertices. Second Figure shows 20 vertices forming a mesh of 12 quadrilaterals.



3.1 Curved Lines and Surfaces

Displays of three dimensional curved lines and surface can be generated from an input set of mathematical functions defining the objects or from a set of user specified data points. When functions are specified, a package can project the defining equations for a curve to

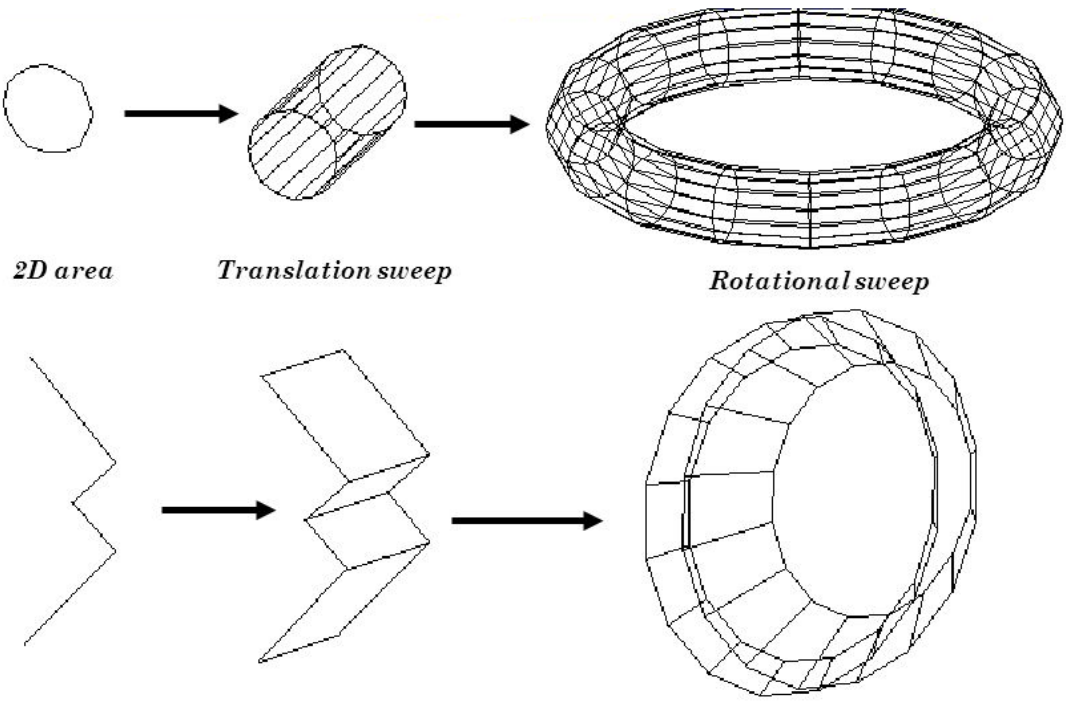
the display plane and plot pixel positions along the path of the projected function. For surfaces, a function is derived to produce a polygon-mesh approximation to the surface.

3.2 Spline Representations

A Spline is a flexible strip used to produce a smooth curve through a designated set of points. Several small weights are distributed along the length of the strip to hold it in proper position on the drafting table as the curve is generated. The Spline curve refers to any sections' curve formed with polynomial sections satisfying specified continuity conditions at the boundary of the pieces. A Spline surface can be described with two sets of orthogonal spline curves. Splines are used in graphics applications to design curve and surface shapes, to digitize drawings for computer storage, and to specify animation paths for the objects or the camera in the scene. It is also used in CAD applications to design the automobiles bodies, aircraft and spacecraft surfaces, and ship hulls.

3.3 Sweep Representations

Sweep representations mean sweeping a 2D surface in 3D space to create an object. However, the objects created by this method are usually converted into polygon meshes and/or parametric surfaces before storing. For this either translational sweep or a rotational sweep is carried out. Following figure shows the same.



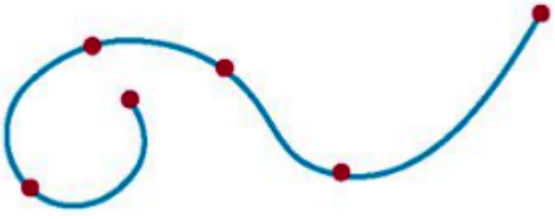
Other variations in sweep representation are:

1. We can specify special path for the sweep as some curve function.
2. We can vary the shape or size of the cross section along the sweep path.
3. We can also vary the orientation of the cross section relative to the sweep path.

3.4 Interpolation and Approximation Splines

Spline curve can be specified by a set of coordinate positions called control points which indicates the general shape of the curve. These control points are fitted with piecewise continuous parametric polynomial functions in one of the two ways:

1. When polynomial sections are fitted so that the curve passes through each control point, the resulting curve is said as an interpolated set of control points. Interpolation curves are used to digitize drawings or to specify animation paths. A set of six control points interpolated with piecewise continuous polynomial sections is shown below:



2. When the polynomials are fitted to the general control point path without necessarily passing through any control points, the resulting curve is said as an approximated set of control points. Approximation curves are used as design tools to structure object surfaces. A set of six control points approximated with piecewise continuous polynomial sections is shown below:

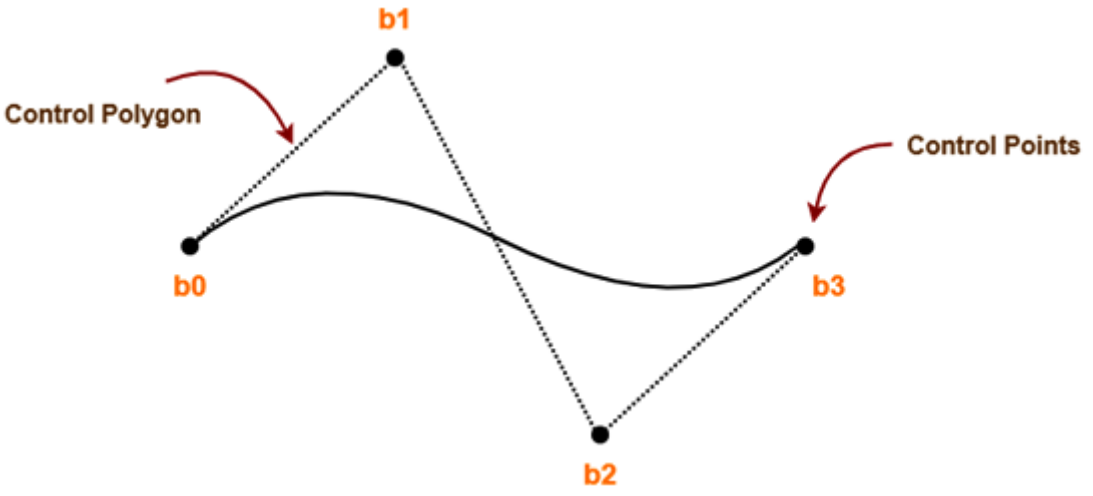
A spline curve is designed, modified and manipulated with operations, specified above, on the control points. The curve can be translated, rotated or scaled with transformation applied to the control points. The convex polygon boundary that encloses a set of control points is called the convex hull. The shape of the convex hull is to imagine a rubber band stretched around the position of the control points so that each control point is either on the perimeter of the hull or inside it. Such a curve is called Bezier Curve.

4. BEZIER CURVE

Bezier curves are used in computer graphics to draw shapes, for Cascaded Style Sheets animation and in many other places.

- Bezier Curve is parametric curve defined by a set of control points.
- Two points are ends of the curve.
- Other points determine the shape of the curve.

The concept of Bezier curves was given by Pierre Bezier. The following curve is an example of a Bezier curve.



Bezier Curve

Here,

1. This Bezier curve is defined by a set of control points b_0, b_1, b_2 and b_3.
2. Points b_0 and b_3 are ends of the curve.
3. Points b_1 and b_2 determine the shape of the curve.

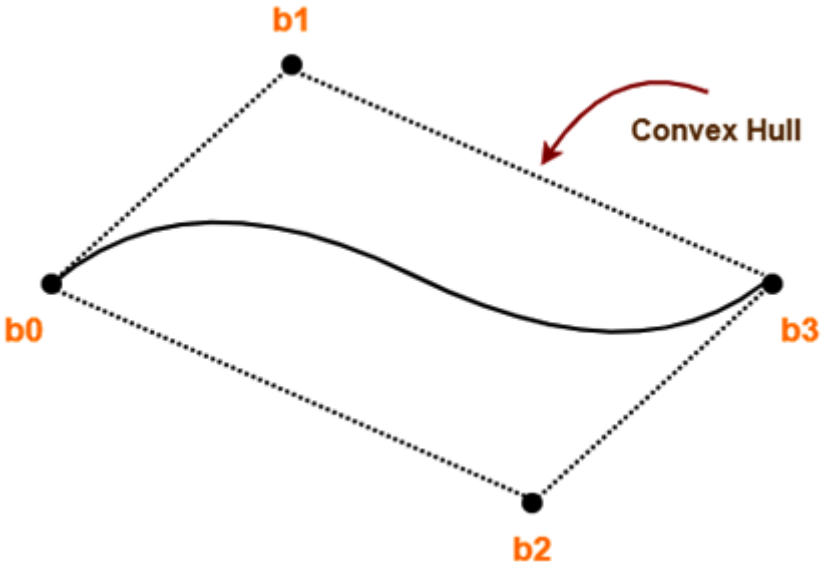
4.1 Properties of Bezier Curves

Few important properties of a Bezier curve are:

1. Bezier curve generally follows the shape of its defining polygon. The first and last points of the curve are coincident with the first and last points of the defining polygon.
2. The degree of the polynomial defining the curve segment is one less than the total number of control points.

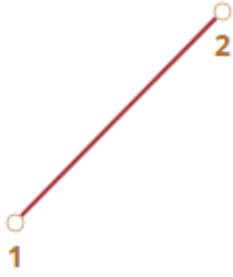
Degree = Number of Control Points – 1

3. The order of the polynomial defining the curve segment is equal to the total number of control points.
4. Bezier curve exhibits the variation diminishing property. It means the curve do not oscillate about any straight line more often than the defining polygon.
5. Bezier curve is always contained within a polygon called as convex hull of its control points shown in below figure:

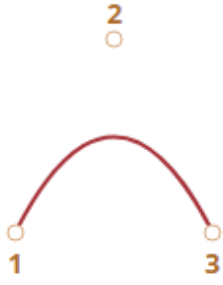


Bezier Curve with Convex Hull

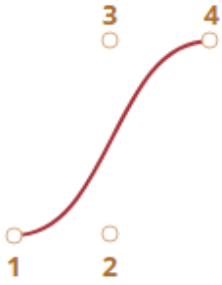
For instance, two points curve:



Three points curve:



Four points curve:



4.2 Applications of Bezier Curves

1. Computer Graphics: Bezier curves are widely used in computer graphics to model smooth curves. The curve is completely contained in the convex hull of its control points. So, the points can be graphically displayed & used to manipulate the curve intuitively.
2. Animation: Bezier curves are used to outline movement in animation applications such as Adobe Flash and Synfig. Users outline the wanted path in Bezier curves. The application creates the needed frames for the object to move along the path. For 3D animation, Bezier curves are often used to define 3D paths as well as 2D curves.
3. Fonts: True type fonts use composite Bezier curves composed of quadratic Bezier curves. Modern imaging systems like Postscript, Asymptote etc. use composite cubic Bezier curves for drawing curved shapes.

Here are some examples:

4.3 Bezier Curve Equation

A Bezier curve is parametrically represented by

$$P(t) = \sum_{0}^{n} B_i \times ({}^{n}C_i \times t^i \times (1-t)^{n-i})$$

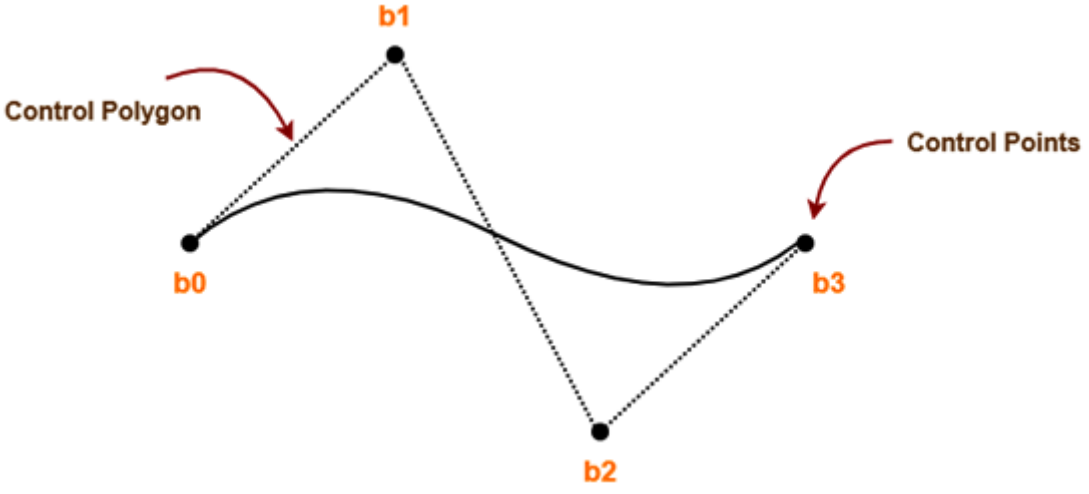
Here,

- P(t) = Any point lying on the Bezier curve
- B_i = i^{th} control point of the Bezier curve
- t is any value between [0,1] i.e. 0 <= t <= 1
- n = degree of the curve

Cubic Bezier Curve

- Cubic Bezier curve is a Bezier curve with degree 3.
- The total number of control points in a cubic Bezier curve is 4.

The following curve is an example of a cubic Bezier curve:



Here,

- This curve is defined by 4 control points b_0, b_1, b_2 and b_3.
- The degree of this curve is 3.
- So, it is a cubic Bezier curve.

Derivation of Cubic Bezier Curve Equation

Substituting n = 3 for a cubic Bezier curve into the equation, we get:

$$P(t) = \sum_{0}^{3} B_i \times ({}^3C_i \times t^i \times (1-t)^{3-i})$$

Expanding the above equation, we get:

$$P(t) = B_0{}^3C_0(t) + B_1{}^3C_1(t) + B_2{}^3C_2(t) + B_3{}^3C_3(t) \quad ...(1)$$

$$^3C_0\,(t) = \frac{3!}{0! \times (3-0)!} \times t^0 \times (1-t)^{3-0}$$

$$= (1\text{-}t)^3 \text{.....(2)}$$

$$^3C_1\,(t) = \frac{3!}{1! \times (3-1)!} \times t^1 \times (1-t)^{3-1}$$

$$= 3t(1\text{-}t)^2 \text{.....(3)}$$

$$^3C_2\,(t) = \frac{3!}{2! \times (3-2)!} \times t^2 \times (1-t)^{3-2}$$

$$= 3t^2(1\text{-}t)^2 \text{.....(4)}$$

$$^3C_3\,(t) = \frac{3!}{3! \times (3-3)!} \times t^3 \times (1-t)^{3-3}$$

$$= t^3 \text{.....(5)}$$

Using (2), (3), (4) and (5) in (1), we get-

$$P(t) = B_0(1\text{-}t)^3 + B_1 3t(1\text{-}t)^2 + B_2 3t^2(1\text{-}t) + B_3 t^3$$

In general, given the coordinates of control points P_i: the first control point has coordinates $P_1 = (x_1, y_1)$, the second: $P_2 = (x_2, y_2)$, and so on. The curve coordinates are described by the equation that depends on the parameter value t from the interval [0,1].

The formula for a 2-points curve:

$$P = (1-t)P_1 + tP_2$$

For 3 control points:

$$P = (1-t)^2P_1 + 2(1-t)tP_2 + t^2P_3$$

For 4 control points:

$$P = (1-t)^3P_1 + 3(1-t)^2tP_2 + 3(1-t)t^2P_3 + t^3P_4$$

Using these vector equations, we can derive new corresponding coordinates x and y by putting values of P(x,y). For instance, the 3-point curve is formed by points (x,y) calculated as:

- $x = (1-t)^2x_1 + 2(1-t)tx_2 + t^2x_3$
- $y = (1-t)^2y_1 + 2(1-t)ty_2 + t^2y_3$

Instead of x_1, y_1, x_2, y_2, x_3, y_3, we put the coordinates of 3 control points, and then as t moves from 0 to 1, for each value of t we will have (x,y) of the curve.

For instance, if control points are (0,0), (0.5, 1) and (1, 0), the equations become:

- $x = (1-t)^2 * 0 + 2(1-t)t * 0.5 + t^2 * 1 = (1-t)t + t^2 = t$
- $y = (1-t)^2 * 0 + 2(1-t)t * 1 + t^2 * 0 = 2(1-t)t = -2t^2 + 2t$

Example: Given a Bezier curve with 4 control points, B_0[1 0], B_1[3 3], B_2[6 3], B_3[8 1], Determine any 5 points lying on the curve. Also, draw a rough sketch of the curve.

Solution:

We have-

- The given curve is defined by 4 control points.
- So, the given curve is a cubic Bezier curve.

The parametric equation for a cubic Bezier curve is-

$$P(t) = B_0(1-t)^3 + B_1 3t(1-t)^2 + B_2 3t^2(1-t) + B_3t^3$$

Substituting the control points B_0, B_1, B_2 and B_3, we get-

$P(t) = [1\ 0](1-t)^3 + [3\ 3]3t(1-t)^2 + [6\ 3]3t^2(1-t) + [8\ 1]t^3$ ……..(1)

Now, To get 5 points lying on the curve, assume any 5 values of t lying in the range 0 <= t <= 1.

Let 5 values of t are 0, 0.2, 0.5, 0.7, 1

For t = 0:

Substituting t=0 in (1), we get-

$P(0) = [1\ 0](1-0)^3 + [3\ 3]3(0)(1-t)^2 + [6\ 3]3(0)^2(1-0) + [8\ 1](0)^3$

$= [1\ 0] + 0 + 0 + 0$

$= [1\ 0]$

For t = 0.2:

Substituting t=0.2 in (1), we get-

$P(0.2) = [1\ 0](1-0.2)^3 + [3\ 3]3(0.2)(1-0.2)^2 + [6\ 3]3(0.2)^2(1-0.2) + [8\ 1](0.2)^3$

$= [1\ 0](0.8)^3 + [3\ 3]3(0.2)(0.8)^2 + [6\ 3]3(0.2)^2(0.8) + [8\ 1](0.2)^3$

$= [1\ 0] \times 0.512 + [3\ 3] \times 3 \times 0.2 \times 0.64 + [6\ 3] \times 3 \times 0.04 \times 0.8 + [8\ 1] \times 0.008$

$= [0.512\ 0] + [1.152\ 1.152] + [0.576\ 0.288] + [0.064\ 0.008]$

$= [2.304\ 1.448]$

For t = 0.5:

Substituting t=0.5 in (1), we get-

$P(0.5) = [1\ 0](1-0.5)^3 + [3\ 3]3(0.5)(1-0.5)^2 + [6\ 3]3(0.5)^2(1-0.5) + [8\ 1](0.5)^3$

$= [1\ 0](0.5)^3 + [3\ 3]3(0.5)(0.5)^2 + [6\ 3]3(0.5)^2(0.5) + [8\ 1](0.5)^3$

$= [1\ 0] \times 0.125 + [3\ 3] \times 3 \times 0.5 \times 0.25 + [6\ 3] \times 3 \times 0.25 \times 0.5 + [8\ 1] \times 0.125$

$= [0.125\ 0] + [1.125\ 1.125] + [2.25\ 1.125] + [1\ 0.125]$

$= [4.5\ 2.375]$

For t = 0.7:

Substituting t=0.7 in (1), we get-

$P(0.7) = [1\ 0](1-0.7)^3 + [3\ 3]3(0.7)(1-0.7)^2 + [6\ 3]3(0.7)^2(1-0.7) + [8\ 1](0.7)^3$

$= [1\ 0](0.3)^3 + [3\ 3]3(0.7)(0.3)^2 + [6\ 3]3(0.7)^2(0.3) + [8\ 1](0.7)^3$

$= [1\ 0] \times 0.027 + [3\ 3] \times 3 \times 0.7 \times 0.09 + [6\ 3] \times 3 \times 0.49 \times 0.3 + [8\ 1] \times 0.343$

$= [0.027\ 0] + [0.567\ 0.567] + [2.646\ 1.323] + [2.744\ 0.343]$

$= [5.984\ 2.233]$

For t = 1:

Substituting t=1 in (1), we get-

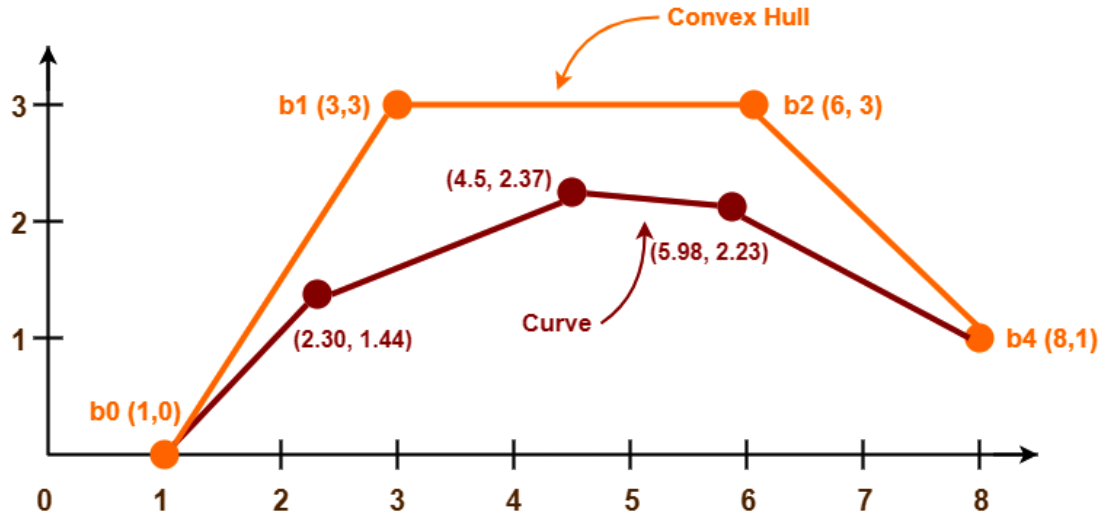
$P(1) = [1\ 0](1-1)^3 + [3\ 3]3(1)(1-1)^2 + [6\ 3]3(1)^2(1-1) + [8\ 1](1)^3$

$= [1\ 0] \times 0 + [3\ 3] \times 3 \times 1 \times 0 + [6\ 3] \times 3 \times 1 \times 0 + [8\ 1] \times 1$

= 0 + 0 + 0 + [8 1]

= [8 1]

Following is the required rough sketch of the curve-



EXERCISE

1. Given a Bezier with four control points $B_0[1,0]$, $B_1[2,2]$, $B_2[6,3]$, $B_3[8,2]$, determine the five points that lie on the curve. Also draw the curve on the graph.
2. Derive equation for quadratic Bezier Curve.

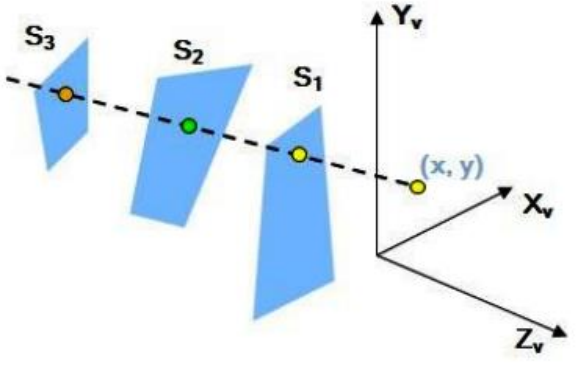
7. VISIBLE SURFACE DETECTION

When we view a picture containing non-transparent objects and surfaces, we cannot see those objects from view which are on the back side of objects. So, we must remove these hidden surfaces to get a realistic screen image. The identification and removal of these surfaces is called Hidden-surface problem. There are two approaches for removing hidden surface problems: Object-Space method and Image-space method. The Object-space method is implemented in physical coordinate system whereas the image-space method is implemented in the screen coordinate system. When we want to display a 3D object on a 2D screen, we need to identify those parts of a screen that are visible from a chosen viewing position. This chapter discusses few methods for visible surface detection, also known as, hidden surface removal.

1. DEPTH BUFFER Z−BUFFER

This method was developed by Cutmull. It is an image-space approach. The basic idea is to test the Z-depth of each surface to determine the closest visible surface. In this method, each surface is processed separately one pixel position at a time across the surface. The depth values for a pixel are compared and the closest smallest z surface determines the color to be displayed in the frame buffer. It is applied very efficiently on surfaces of polygon. Surfaces can be processed in any order. To override the closer polygons from the far ones, two buffers, frame buffer and depth buffer, are used. Depth buffer is used to store depth values for (x,y) position, as surfaces are processed $0 \leq depth \leq 1$.

The frame buffer is used to store the intensity value of color value at each position (x,y). The z-coordinates are usually normalized to the range [0, 1]. The 0 value for z-coordinate indicates back clipping pane and 1 value for z-coordinates indicates front clipping pane.



Algorithm

1. Set the buffer values

   DepthBuffer(x,y) = 0

   FrameBuffer(x,y) = backgroundColor

2. Process each polygon One at a time

   For each projected (x,y) pixel position of a polygon, calculate depth z.

   If Z > DepthBuffer(x,y)

   Compute surfaceColor,

   set DepthBuffer(x,y) = z,

   FrameBuffer(x,y) = surfaceColor(x,y)

Advantages

- It is easy to implement.
- It reduces the speed problem if implemented in hardware.
- It processes one object at a time.

Disadvantages

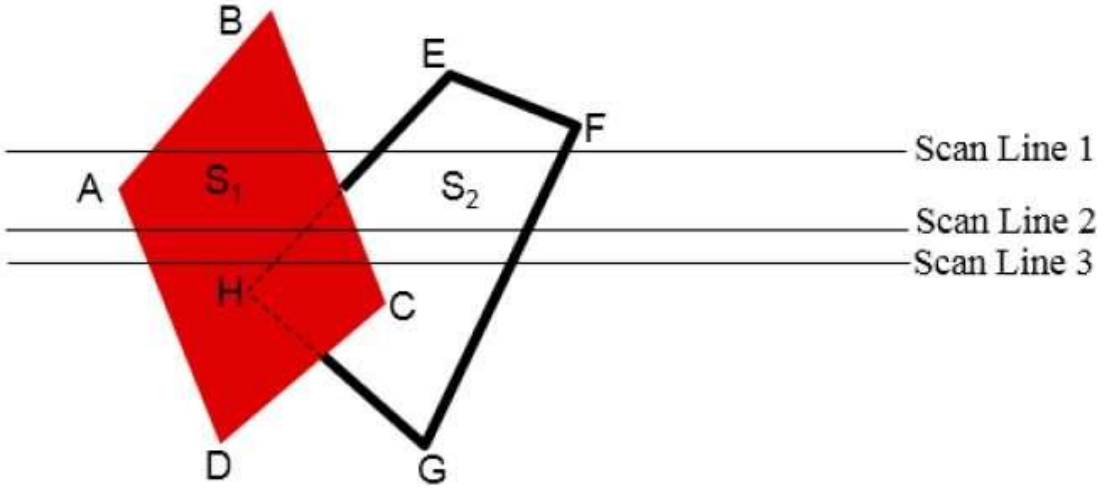
- It requires large memory.
- It is time consuming process.

2. SCAN-LINE METHOD

It is an image-space method to identify visible surface. This method has a depth information for only single scan-line. In order to acquire one scan-line of depth values, we must group and process all polygons intersecting a given scan-line at the same time before processing the next scan-line. Two important tables, edge table and polygon table, are maintained for this, as we discussed in chapter 6.

The Edge Table − It contains coordinate endpoints of each line in the scene, the inverse slope of each line, and pointers into the polygon table to connect edges to surfaces.

The Polygon Table − It contains the plane coefficients, surface material properties, other surface data, and may be pointers to the edge table.

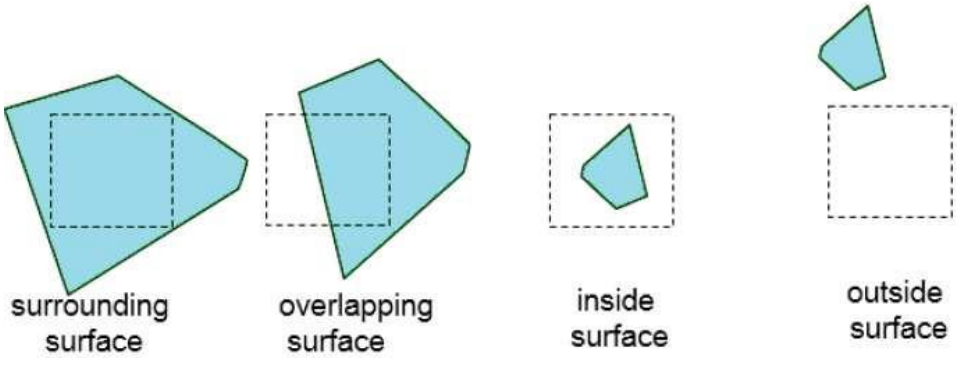


To facilitate the search for surfaces crossing a given scan-line, an active list of edges is formed. The active list stores only those edges that cross the scan-line in order of increasing x value. Also a flag is set for each surface to indicate whether a position along a scan-line is either inside or outside the surface. Pixel positions across each scan-line are processed from left to right. At the left intersection with a surface, the surface flag is turned on and at the right, the flag is turned off. We only need to perform depth calculations when multiple surfaces have their flags turned on at a certain scan-line position.

3. AREA-SUBDIVISION METHOD

The area-subdivision method takes advantage by locating those view areas that represent part of a single surface. It divides the total viewing area into smaller and smaller rectangles until each small area is the projection of part of a single visible surface or no surface at all. Continue this process until the subdivisions are easily analyzed as belonging to a single surface or until they are reduced to the size of a single pixel. An easy way to do this is to successively divide the area into four equal parts at each step. There are four possible relationships that a surface can have with a specified area boundary.

- Surrounding surface − One that completely encloses the area.
- Overlapping surface − One that is partly inside and partly outside the area.
- Inside surface − One that is completely inside the area.
- Outside surface − One that is completely outside the area.



The tests for determining surface visibility within an area can be stated in terms of these four classifications. No further subdivisions of a specified area are needed if one of the following conditions is true:

- All surfaces are outside surfaces with respect to the area.
- Only one inside, overlapping or surrounding surface is in the area.
- A surrounding surface obscures all other surfaces within the area boundaries.

4. BACK-FACE DETECTION

A fast and simple object-space method for identifying the back faces of a polyhedron is based on the inside-outside tests. A point (x,y,z) is inside a polygon surface with plane parameters A, B, C, and D if an inside point is along the line of sight to the surface. The

polygon must be a back face and we are inside that face and cannot see the front of it from our viewing position. We can simplify this test by considering the normal vector N to a polygon surface, which has Cartesian components A,B,C. In general, if V is a vector in the viewing direction from the eye or "camera" position, then this polygon is a back face if

$$V.N > 0$$

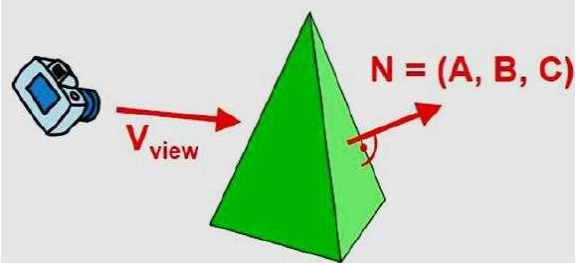
Furthermore, if object descriptions are converted to projection coordinates and your viewing direction is parallel to the viewing z-axis, then

$$V = (0, 0, V_z) \text{ and } V.N = V_zC$$

So that we only need to consider the sign of C in the component of the normal vector N.

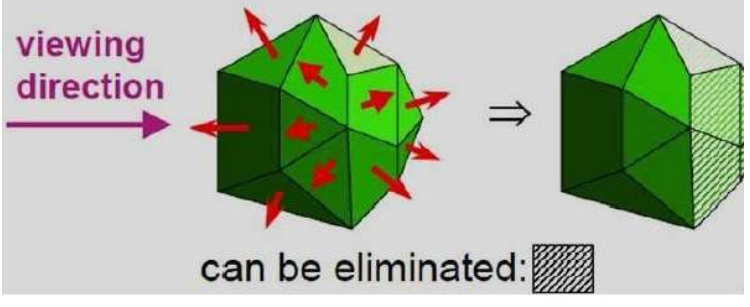
In a right-handed viewing system with viewing direction along the negative Z_v axis, the polygon is a back face if $C < 0$. Also, we cannot see any face whose normal has z component $C = 0$, since our viewing direction is towards that polygon. Thus, in general, we can label any polygon as a back face if its normal vector has a z component value,

$$C <= 0$$



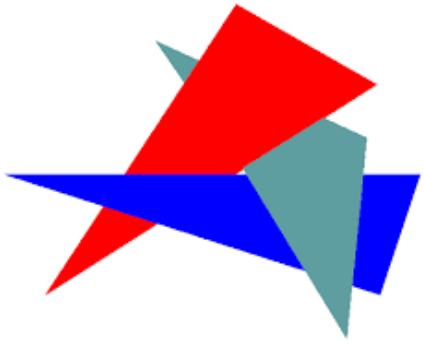
Similar methods can be used in packages that employ a left-handed viewing system. In these packages, plane parameters A, B, C and D can be calculated from polygon vertex coordinates specified in a clockwise direction unlike the counter clockwise direction used in a right handed system.

Also, back faces have normal vectors that point away from the viewing position and are identified by $C >= 0$ when the viewing direction is along the positive Z_v axis. By examining parameter C for the different planes defining an object, we can immediately identify all the back faces.



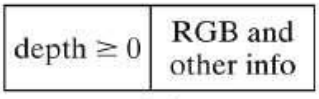
5. A-BUFFER METHOD

The A-buffer method is an extension of the depth-buffer method. The A-buffer method is a visibility detection method developed at Lucas film Studios for the rendering system Renders Everything You Ever Saw (REYES). The A-buffer expands on the depth buffer method to allow transparencies. The key data structure in the A-buffer is the accumulation buffer.

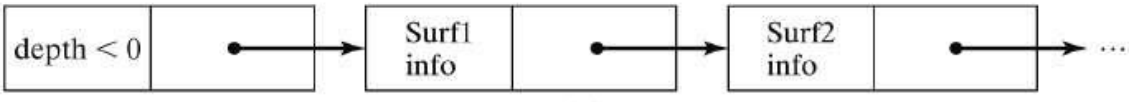


Each position in the A-buffer has two fields:

- Depth field − It stores a positive or negative real number
- Intensity field − It stores surface-intensity information or a pointer value



(a)



(b)

1. If depth >= 0, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area. The intensity field then stores the RGB components of the surface color at that point and the percent of pixel coverage.
2. If depth < 0, it indicates multiple-surface contributions to the pixel intensity. The intensity field then stores a pointer to a linked list of surface data. The surface buffer in the A-buffer includes:
   - RGB intensity components
   - Opacity Parameter
   - Depth
   - Percent of area coverage

- Surface identifier

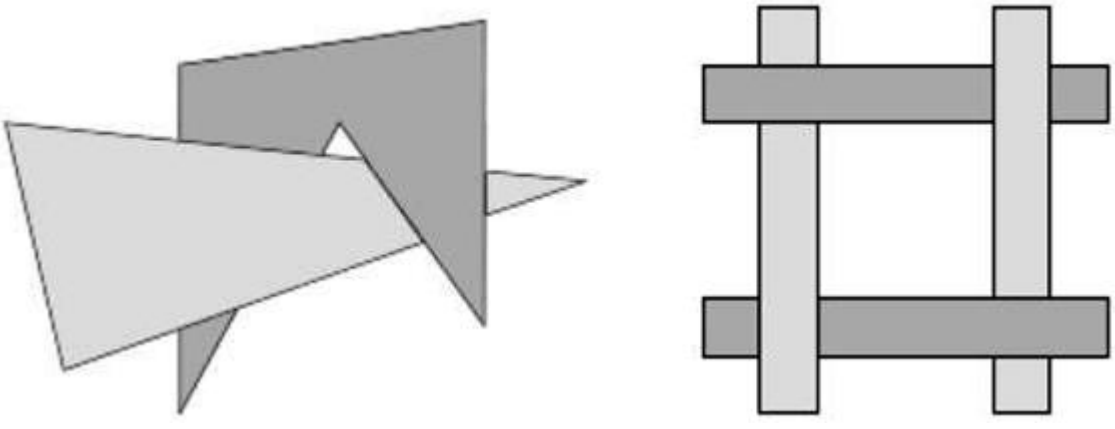
The algorithm proceeds just like the depth buffer algorithm. The depth and opacity values are used to determine the final color of a pixel.

6. DEPTH SORTING METHOD

Depth sorting method uses both image space and object-space operations. The depth-sorting method performs two basic functions:

- First, the surfaces are sorted in order of decreasing depth.
- Second, the surfaces are scan-converted in order, starting with the surface of greatest depth.

The scan conversion of the polygon surfaces is performed in image space. This method, for solving the hidden-surface problem, is often referred to as the painter's algorithm. The following figure shows the effect of depth sorting:



The algorithm begins by sorting by depth. For example, the initial depth estimate of a polygon may be taken to be the closest z value of any vertex of the polygon.

Let us take the polygon P at the end of the list. Consider all polygons Q whose z-extents overlap P's. Before drawing P, we make the following tests. If any of the following tests is positive, then we can assume P can be drawn before Q.

- Do the x-extents not overlap?
- Do the y-extents not overlap?
- Is P entirely on the opposite side of Q's plane from the viewpoint?
- Is Q entirely on the same side of P's plane as the viewpoint?
- Do the projections of the polygons not overlap?

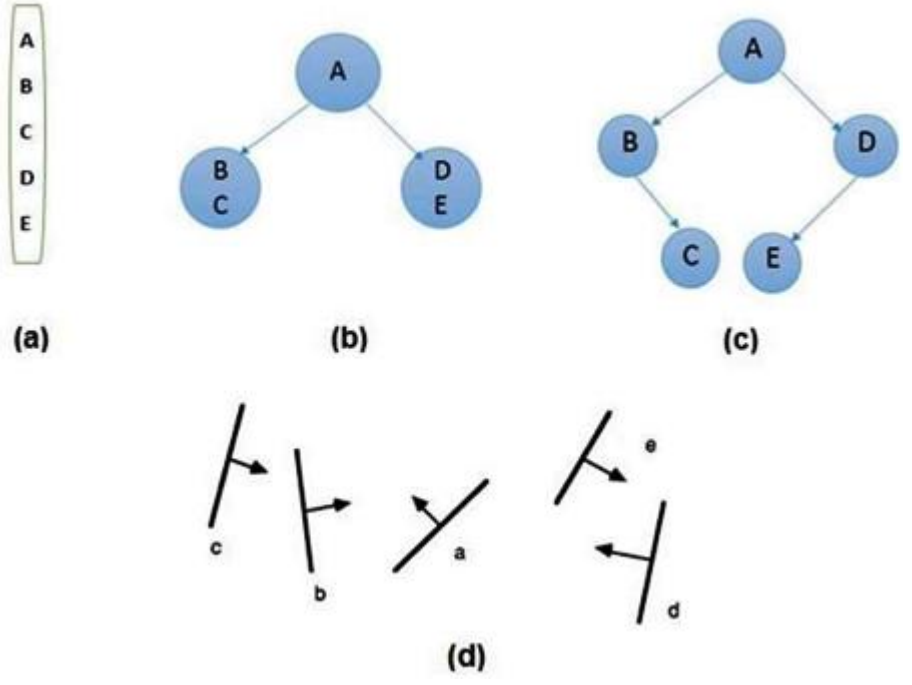
If all the tests fail, then we split either P or Q using the plane of the other. The new cut polygons are inserted into the depth order and the process continues. Theoretically, this partitioning could generate $O(n^2)$ individual polygons, but in practice, the number of polygons is much smaller.

7. BINARY SPACE PARTITION - BSP TREES

Binary space partitioning is used to calculate visibility. To build the BSP trees, follow the steps given below:

1. Starts with polygons and label all the edges.
2. Dealing with only one edge at a time, extend each edge so that it splits the plane into two.
3. Place the first edge in the tree as root.
4. Add subsequent edges based on whether they are inside or outside.
5. Edges that span the extension of an edge that is already in the tree are split into two and both are added to the tree.

Example:



1. From the above figure, first take A as a root.
2. Make a list of all nodes in figure aa.
3. Put all the nodes that are in front of root A to the left side of node A and put all those nodes that are behind the root A to the right side as shown in figure bb.
4. Process all the front nodes first and then the nodes at the back.
5. As shown in figure cc, we will first process the node B. As there is nothing in front of the node B, we have put NIL. However, we have node C at back of node B, so node C will go to the right side of node B.
6. Repeat the same process for the node D.

8. BASICS OF IMAGE PROCESSING

The digital image processing deals with developing a digital system that performs operations on a digital image. An image is nothing more than a two dimensional signal. It is defined by the mathematical function f(x,y) where x and y are the two co-ordinates horizontally and vertically and the amplitude of f at any pair of coordinate (x, y) is called the intensity or gray level of the image at that point. When (x, y) and the amplitude values of f are all finite discrete quantities, we call the image a digital image. A digital image is composed of a finite number of elements, each of which has a particular location and values of these elements are referred to as picture elements, image elements and pixels. The field of digital image processing refers to the processing of digital image by means of a digital computer.

1. COMPONENTS OF IMAGE PROCESSING SYSTEM

1 **Image Sensors:** With reference to sensing, two elements are required to acquire digital image. The first is a physical device that is sensitive to the energy radiated by the object of which we want to capture the image and second is specialized image processing hardware.

2 **Specialize image processing hardware:** It consists of the digitizer just mentioned, plus hardware that performs other primitive operations such as addition, subtraction and logical operations on images.

3 **Computer:** It is a general purpose computer and can range from a PC to a supercomputer depending on the application requirements. In dedicated applications, sometimes specially designed computer are used to achieve a required level of performance.

4 **Software:** It consists of specialized modules that perform specific tasks with a well designed package which includes capability for the user to write and execute code.

5 **Mass storage:** This capability is a must in an image processing applications. For example, an image of size 1024X1024 pixels, in which the intensity of each pixel is an 8-bit quantity requires one MB of storage space if the image is not compressed.

6 **Image display:** Image displays, in use today, are mainly color TV monitors. These monitors are driven by the outputs of image and graphics displays cards that are an integral part of computer system.

7 **Hardcopy devices:** The devices for recording image includes laser printers, film cameras, heat sensitive devices, inkjet units and digital units such as optical and CD ROM disks. Films provide the highest possible resolution, but paper is the obvious medium of choice for written applications.

8 **Networking:** It is almost a default function in any computer system in use today because of the large amount of data transfer required in image processing applications. The key consideration in image transmission bandwidth which affects the data transfer.

2. IMAGE FORMATION

Images are formed in the similar way the human eye forms the image. First, let us understand the how does the human eye work.

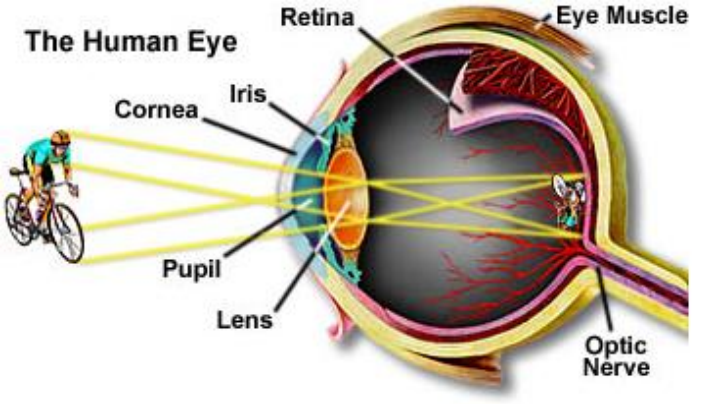
2.1 Structure of Human eye

The eye is nearly a sphere with average approximately 20 mm diameter. The eye is enclosed with three membranes:

1. The cornea and sclera – It is a tough, transparent tissue that covers the anterior surface of the eye. Rest of the optic globe is covered by the sclera.

2. The choroid – It contains a network of blood vessels that serves as the major source of nutrition to the eyes. It helps to reduce extraneous light entering in the eye. It has two parts:

   a. Iris Diaphragms – It contracts or expands to control the amount of light that enters the eyes
   b. Ciliary body – The shape of the lens of the eye is controlled by tension in the fiber of the ciliary body.



Human Eye

3. Retina – It is innermost membrane of the eye. When the eye is properly focused, light from an object outside the eye is imaged on the retina. There are various light receptors over the surface of the retina. The two major classes of the receptors are:

   a. Cones – It is about 6 to 7 million in number. These are located in the central portion of the retina called the fovea. They are highly sensitive to color. Human can resolve fine details with these cones because each one is connected to its own nerve end. Cone vision is called photonic or bright light vision.
   b. Rods – These are very much in number ranging from 75 to 150 million and are distributed over the entire retinal surface. The large area of distribution and the fact that several roads are connected to a single nerve give a general overall picture of the field of view. They are not involved in the color vision and are sensitive to low level of illumination. Rod vision is called isotopic or dim light vision. The absence of these reciprocators is called blind spot.

2.2 Image formation in the eye

The major difference between the lens of the eye and an ordinary optical lens is that the former is more flexible. As stated earlier, the shape of the lens of the eye is controlled by

tension in the fiber of the ciliary body. To focus on the distant object, the controlling muscles allow the lens to become thicker in order to focus on object near the eye and thus it becomes relatively flattened. The distance between the center of the lens and the retina is called the focal length and it varies from 17mm to 14mm as the refractive power of the lens increases from its minimum to its maximum**.** When the eye focuses on an object farther away than about 3m. the lens exhibits its lowest refractive power. When the eye focuses on a nearly object, the lens is most strongly refractive. The retinal image is reflected primarily in the area of the fovea. Then the perception takes place by the relative excitation of light receptors, which transform radiant energy into electrical impulses that are ultimately decoded by the brain.

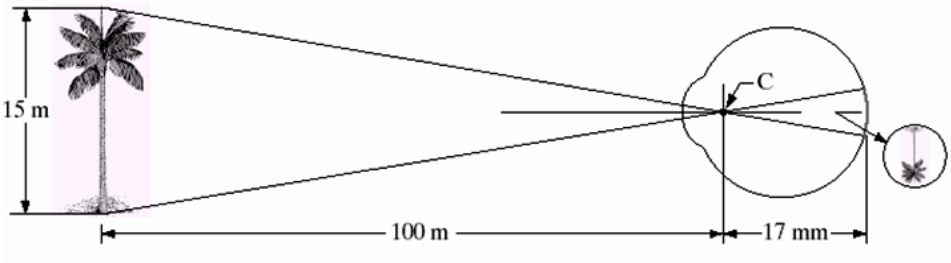
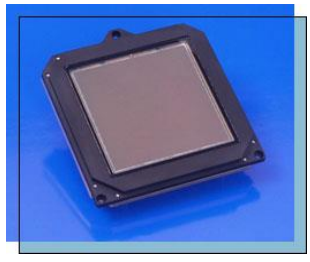


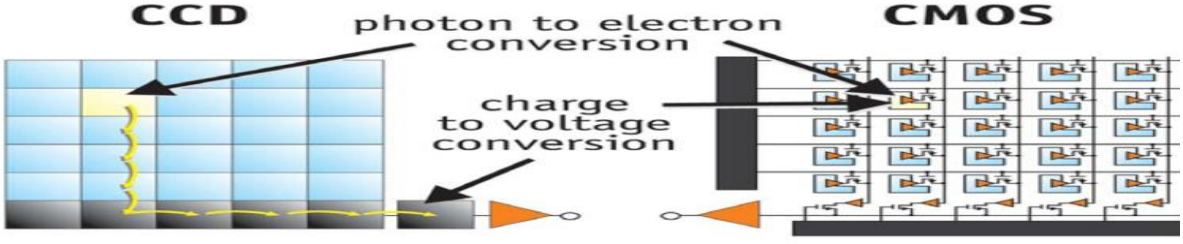
Image formation in Human eye

2.3 Image Formation using cameras



Digital camera replaces traditional film with a sensor array. Each cell in the array is light-sensitive diode that converts photons to electrons. There are two common types of digital cameras:

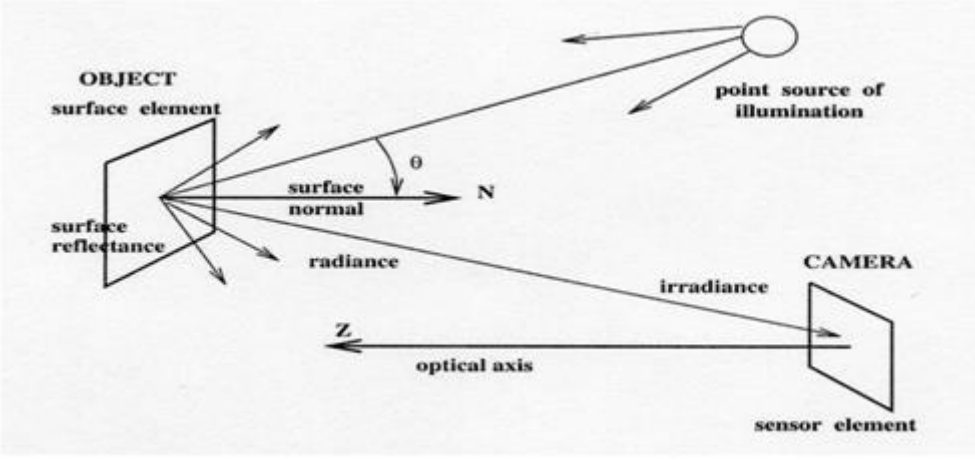
1. Charge Coupled Device (CCD)
2. Complementary metal oxide semiconductor (CMOS)



CCD Vs CMOS Camera

CCDs move photo generated charge from pixel to pixel and convert it to voltage at an output node. An analog-to-digital converter (ADC) then turns each pixel's value into a

digital value. However, CMOS Cameras convert charge to voltage inside each element. They use several transistors at each pixel to amplify and move the charge using more traditional wires. Since the CMOS signal is digital, so it does not need ADC. To form the image through camera, the scene is illuminated by a single source. Then scene reflects radiation towards the camera and the camera senses it via solid state cells.



The image formation model

There are two parts to the image formation process:

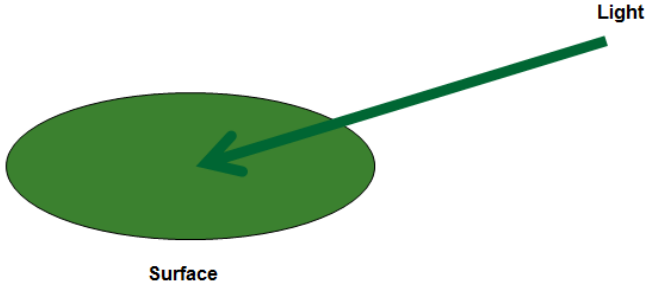
(1) The geometry, which determines wherein the image plane i.e. the projection of a point in the scene will be located.

(2) The physics of light, which determines the brightness of a point in the image plane.

The irradiance at the image pixel is converted into the brightness of the pixel. Image Irradiance is proportional to Scene Radiance. Scene distance i.e. distance between object and viewer z, does not affect image brightness. The angle of the scene patch with respect to the view(α) reduces the brightness by the $\cos^4\alpha$. In practice the effect is even stronger.

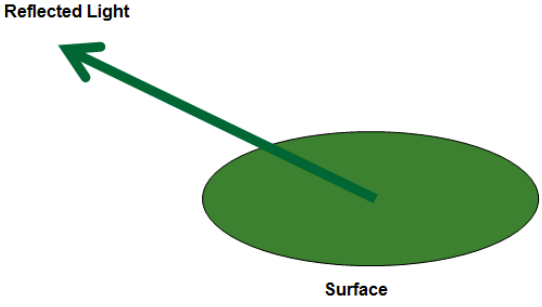
To understand this image formation, we have to learn few terminologies given below:

Irradiance (E): Irradiation is where the radiation is falling on the surface. It is calculated in light power per unit area (watts per square meter) incident on a surface and the apparent extension of the edges of an illuminated object seen against a dark background.



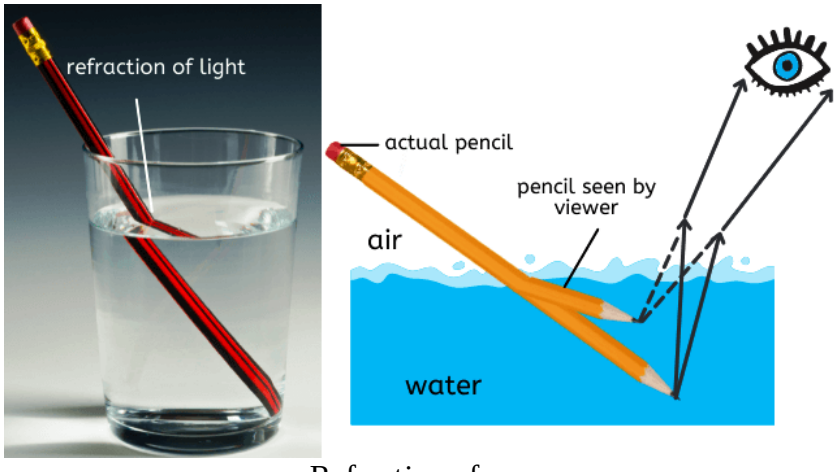
Light incidence to surface

Radiance (L): Radiation is the number of photons that are being emitted by a single source. Amount of light radiated from a surface into a given solid angle per unit area is given in watts per square meter per steradian.



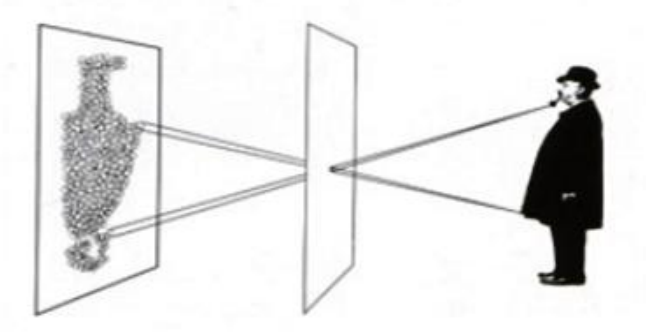
Refelction

Refraction: The change in direction of a wave passing from one medium to another caused by its change in speed due to density of the medium.

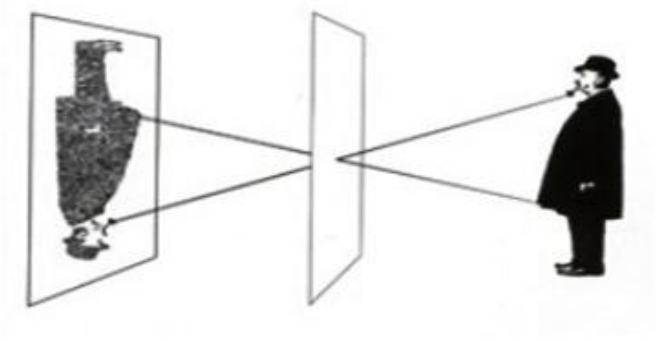


Refraction of rays

Aperture: Aperture refers to the opening of a lens's diaphragm through which light passes. The large aperture make image blurry as light from the source spreads across the image (i.e., not properly focused) whereas smaller aperture size reduces blurring but it limits the amount of light entering the camera and causes light diffraction.

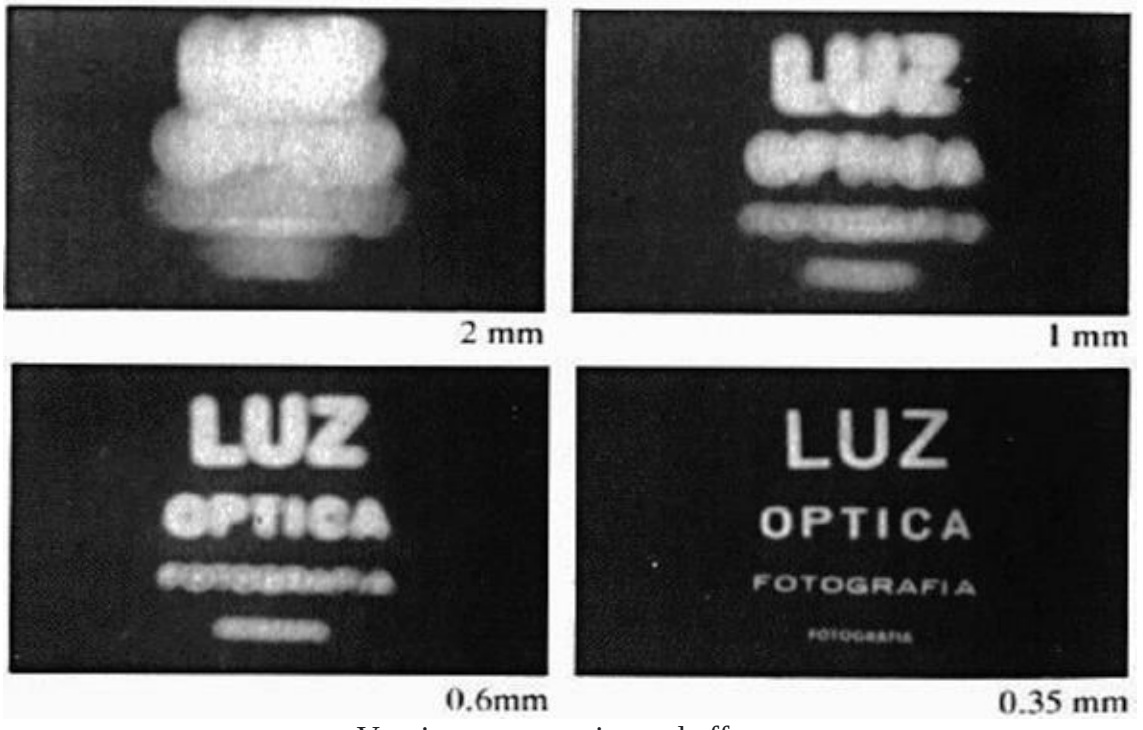


Larger aperture size



Smaller aperture size

Following figure shows varying aperture sizes and its effect on image formation.



Varying aperture size and effects

2.4 Lens

Lens duplicates pinhole geometry without resorting to undesirably small apertures. It gathers all the light radiating from an object point towards the lens's finite aperture and bring light into focus at a single distinct image point. Lens improves image quality, leading to sharper images.

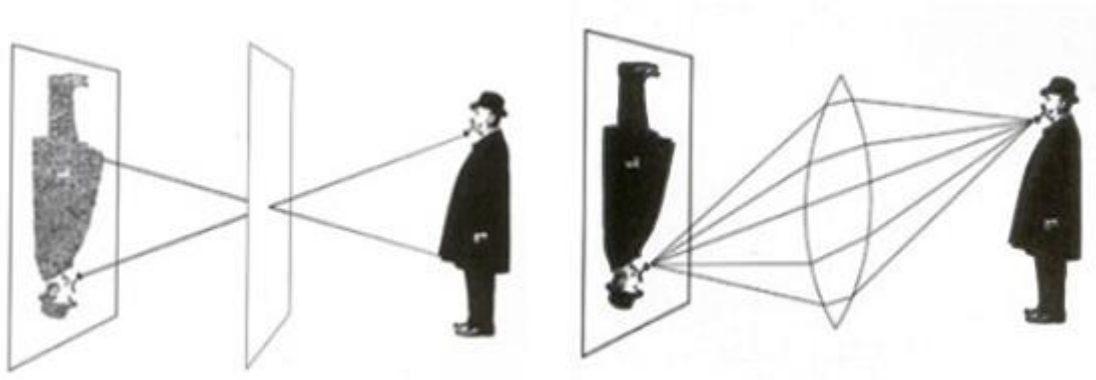
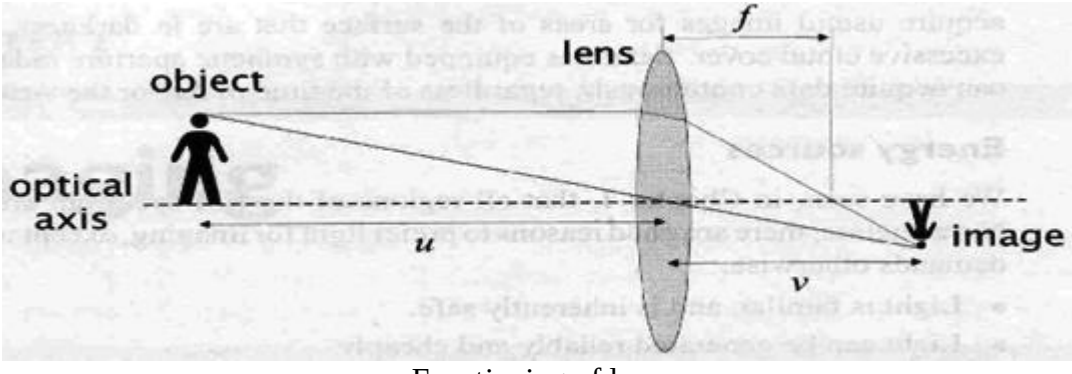


Image formation without lens Image formation using lens

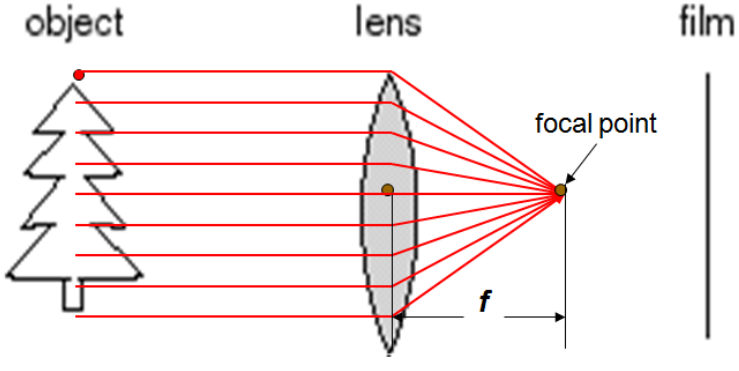
Following figure shows functioning of lens.



Functioning of lens

The lens has following properties which makes it more useful in digital cameras.

1. Light rays passing through the center are not deviated.
2. Light rays passing through a point far away from the center are deviated more.
3. All parallel rays converge to a single point.
4. When rays are perpendicular to the lens, it is called focal point.
5. The plane parallel to the lens at the focal point is called the focal plane.
6. The distance between the lens and the focal plane is called the focal length (f) of the lens.



3. DIGITAL IMAGE:

A digital image is denoted by a two dimensional function of the form f(x,y). The value or amplitude of f at spatial coordinates (x,y) is a positive scalar quantity whose physical meaning is determined by the source of the image. When an image is generated by a physical process, its values are proportional to energy radiated by a physical source. As a consequence, f(x,y) must be nonzero and finite.

The function f(x,y) may be characterized by two components:

1. The amount of the source illumination incident on the scene being viewed.
2. The amount of the source illumination reflected back by the objects in the scene

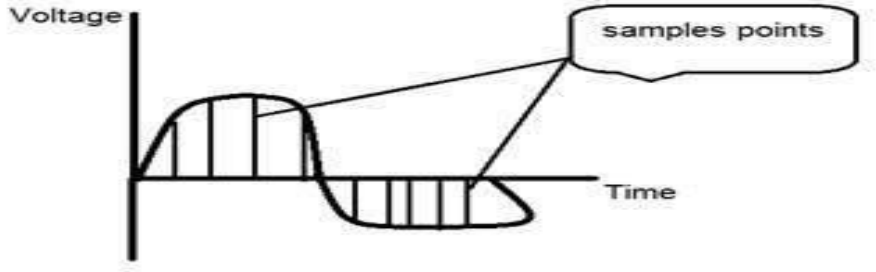
These are called illumination and reflectance components and are denoted by i(x,y) and r(x,y) respectively.

$$f(x,y) = r(x,y) * i(x,y)$$

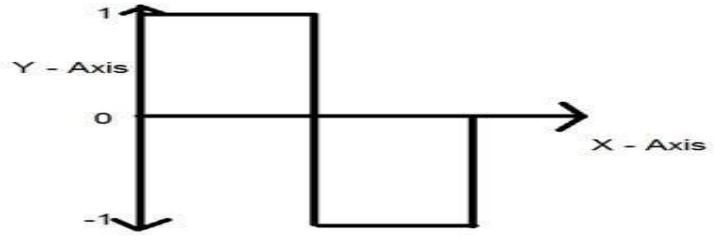
where, Reflectance value ranges in [0,1] and illumination value vary in interval [0,inf].

3.1 Image Sampling And Quantization

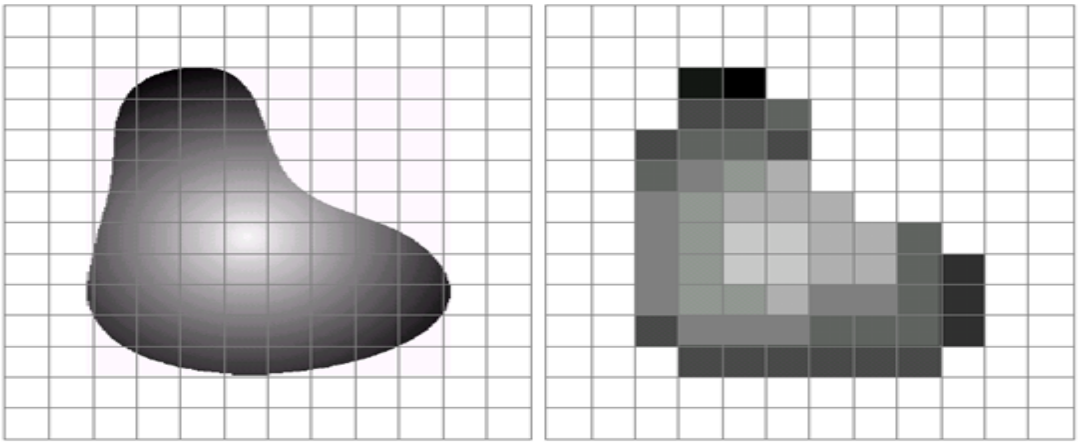
To create a digital image, we need to convert the continuous signal into digital from. This involves two processes – sampling and quantization. An image, f(x, y), that we want to convert to digital form will be continuous with respect to the x- and y-coordinates, and also in amplitude. To convert it to digital form, we have to sample the function in both coordinates and in amplitude. So the part that deals with the digitizing of coordinates is known as sampling and the part that deals with digitizing the amplitude is known as quantization.



Sampling



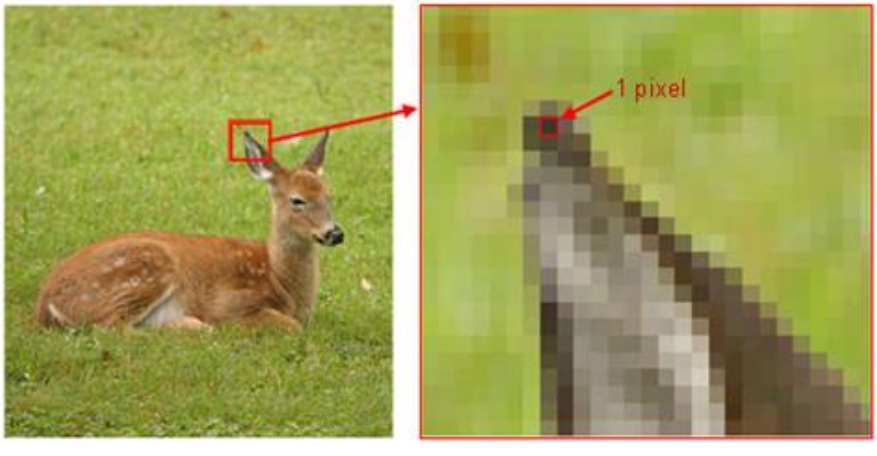
Quantization



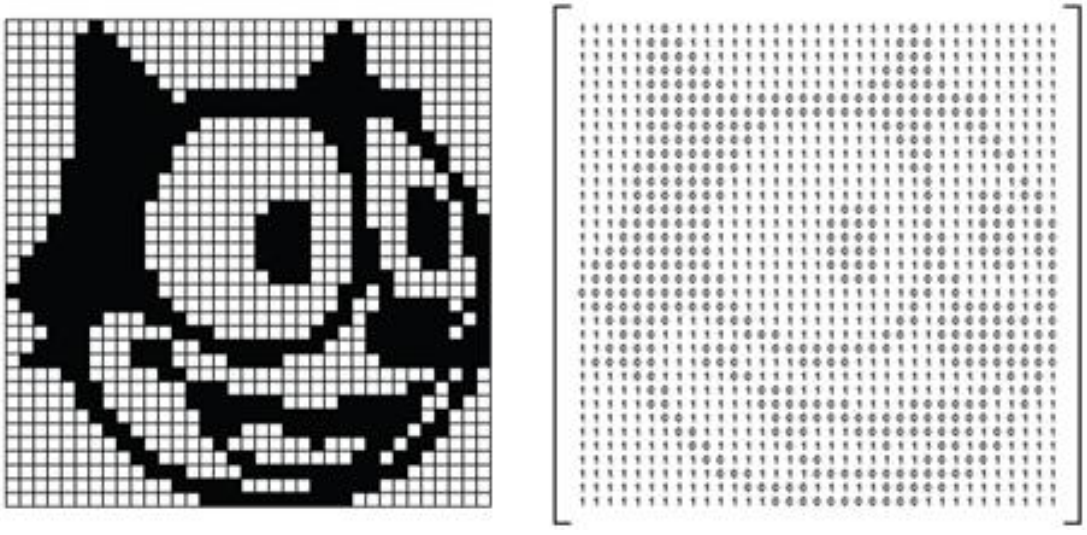
A digital image f(m,n) described in a 2D discrete space is derived from an analog image f(x,y) in a 2D continuous space through a sampling process that is frequently referred to as digitization. The mathematical representation of the image is as follow:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & f(1,2) & \dots & f(1,N-1) \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ f(M-1,0) & f(M-1,1) & f(M-1,2) & \dots & f(M-1,N-1) \end{bmatrix}$$

The 2D continuous image f(m,n) is divided into N rows and M columns. The intersection of a row and a column is termed a pixel. The value assigned to the integer coordinates (m,n) with m=0,1,2..N-1 and n=0,1,2…N-1. In fact, in most cases, f(m,n) is actually a function of many variables including depth, color and time (t). The effect of digitization after sampling and quantization is shown in figure below.

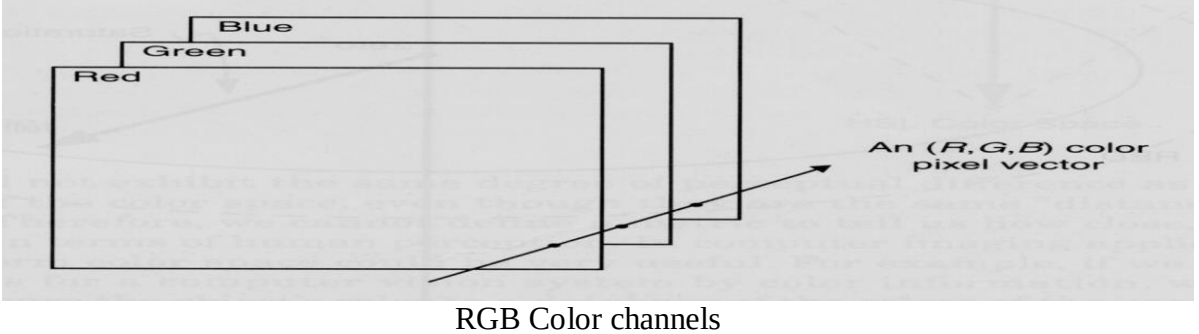


Ultimately, an image will be represented, as shown in the figure below, in the form of a f(m,n) matrix as stated earlier.



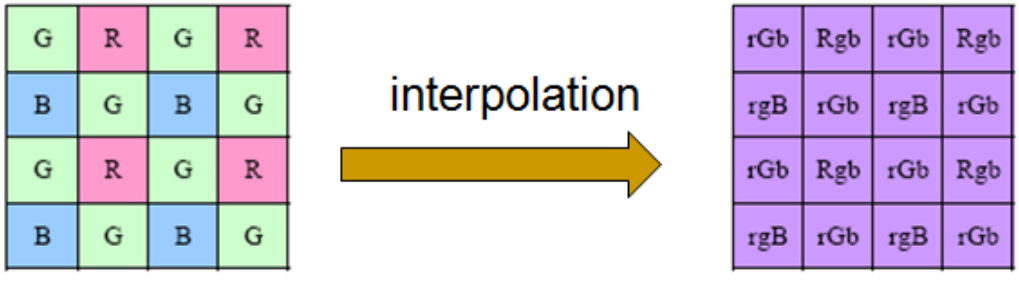
3.2 Color Images

Color images are comprised of three color channels – red, green, and, blue which are combined to create most of the colors a human eye can see.

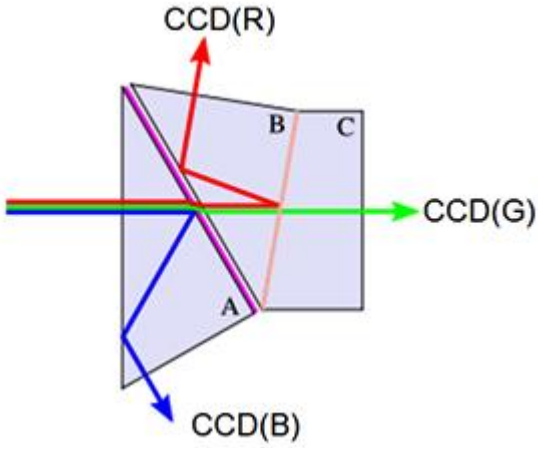


RGB Color channels

The Color sensing in camera is done through a digital prism which requires three chips and precise alignment. It also uses color filter array for sensing the colors.



RGB Color Array



Color sensing prism

3.3 Alternative Color Spaces

Color images use various color spaces which can be computed from RGB channels. This can be done for:

1. Decorrelating the color channels.
2. Bringing color information to the front or core.
3. Perceptual uniformity.

Following various color spaces are used in digital images:

1. RGB (CIE-International Commission on Illumination), RnGnBn (TV - National Television Standard Committee)
2. XYZ (CIE) - The CIE XYZ color space encompasses all color sensations that are visible to a person with average eyesight. That is why CIE XYZ (Tristimulus values) is a device-invariant representation of color. It serves as a standard reference against which many other color spaces are defined.
3. UVW - Wyszecki invented the UVW color space in order to be able to calculate color differences without having to hold the luminance constant. (UCS de la CIE), U*V*W* (UCS modified by the CIE)
4. YUV space uses mathematical formulas to represent colors. In this case, Y varies from 0 to 1 or 0 to 255 if you want to get technical with digital formats. U and V range from -0.5 to 0.5
5. HSV HSV (which stands for Hue Saturation Value) scale provides a numerical readout of your image that corresponds to the color names contained therein. Hue is measured in degrees from 0 to 360. For instance, cyan falls between 181–240 degrees, and magenta falls between 301–360 degrees
6. YIQ is the color space used by the analog NTSC color TV system, employed mainly in North and Central America, and Japan. Y stands for luminance part and IQ stands for chrominance part. I stands for in-phase, while Q stands for quadrature, referring to the components used in quadrature amplitude modulation. It is the most widely colour model used in Television broadcasting.

7. YCbCr (also Y'CbCr, Y Pb/Cb Pr/Cr) is a family of color spaces used in videos and digital photography that separates brightness (luma) from chroma (color).
8. YDbDr used in analog terrestrial color TVs.
9. DSH, HLS, IHS, Munsel color space (cylindrical representation), CIELuv, CIELab, SMPTE-C RGB, YES (Xerox), Kodak Photo CD, YCC, YPbPr etc.

3.4 Image file formats

Many image formats adhere to the simple model shown below to store the image data i.e. pixel values (line by line, no breaks between lines).

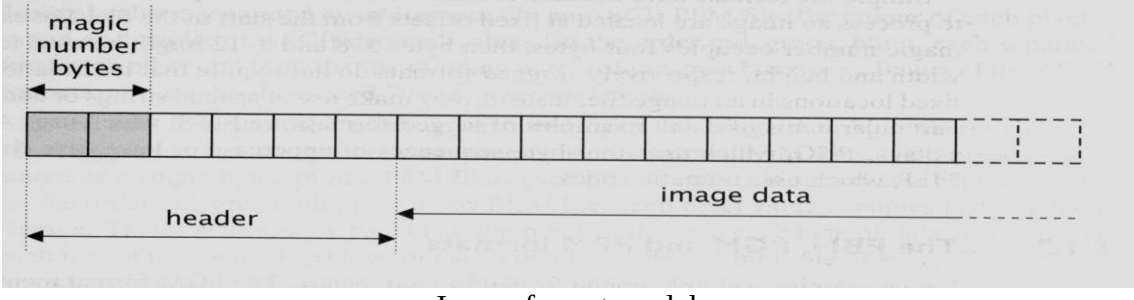


Image format model

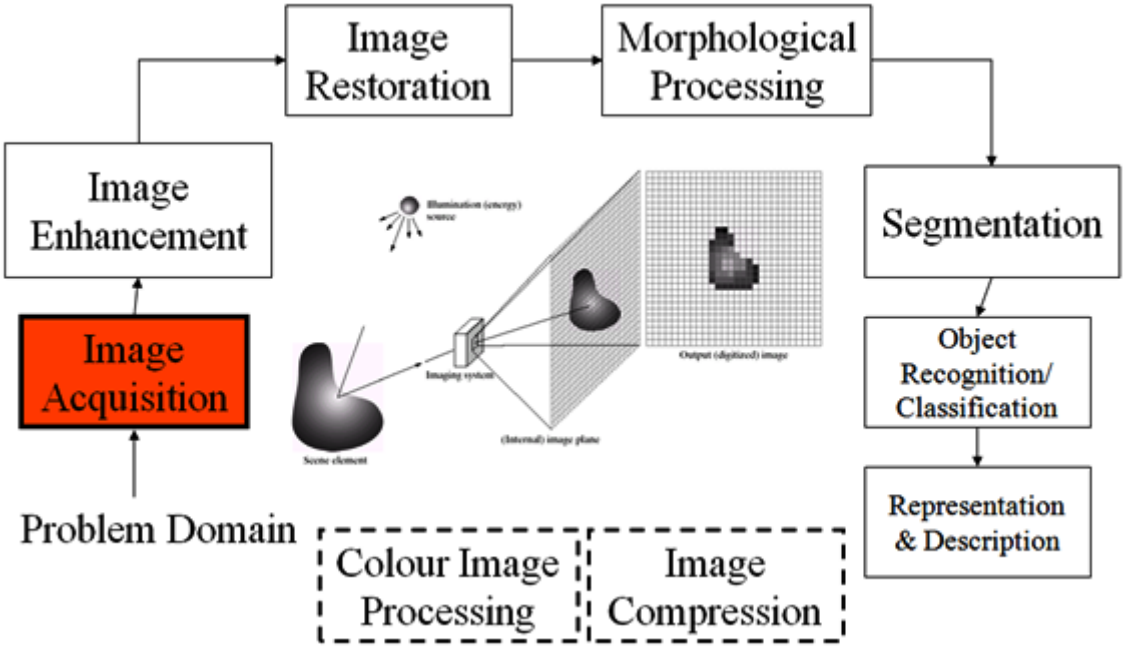
The header contains at least the width and height of the image. Most headers begin with a signature or "magic number" (i.e. a short sequence of bytes for identifying the file format). Based on these, the various image formats have been created which are given below:

1. GIF (Graphic Interchange Format) -
2. PNG (Portable Network Graphics)
3. JPEG (Joint Photographic Experts Group)
4. TIFF (Tagged Image File Format)
5. PGM (Portable Gray Map)
6. FITS (Flexible Image Transport System)
7. A popular format for grayscale images (8 bits/pixel)
8. Closely-related formats are:
    a. PBM (Portable Bitmap), for binary images (1 bit/pixel)
    b. PPM (Portable Pixelmap), for color images (24 bits/pixel)

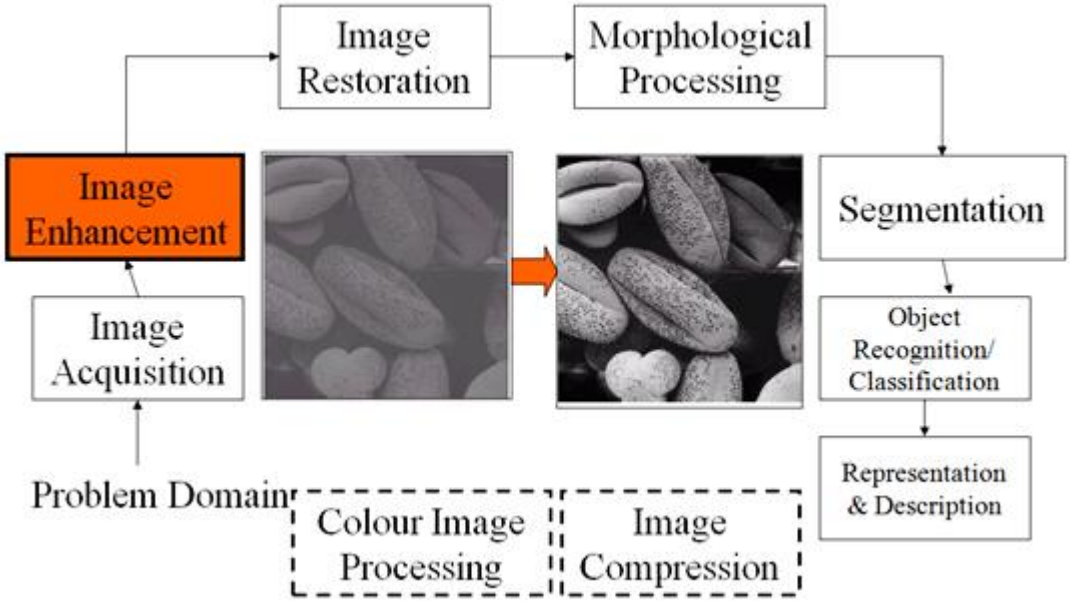
4. STEPS IN IMAGE PROCESSING

Image processing is a sequence of many steps from capturing the image to making a decision. Following are stage in a digital image processing.

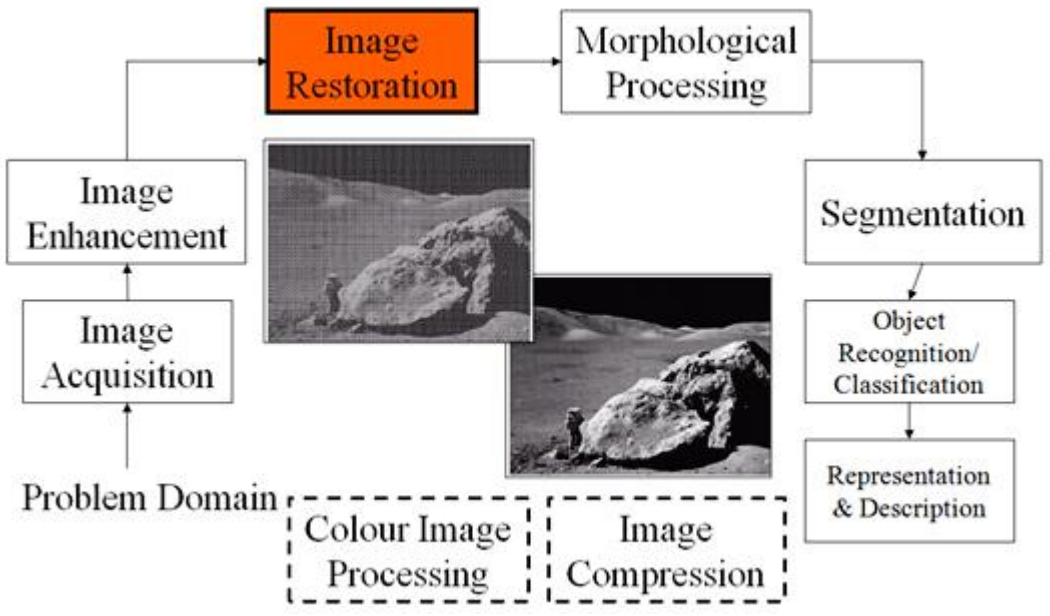
1. **Image acquisition:** It could be as simple as being given an image that is already in digital form. Generally the image acquisition stage involves processing such scaling.



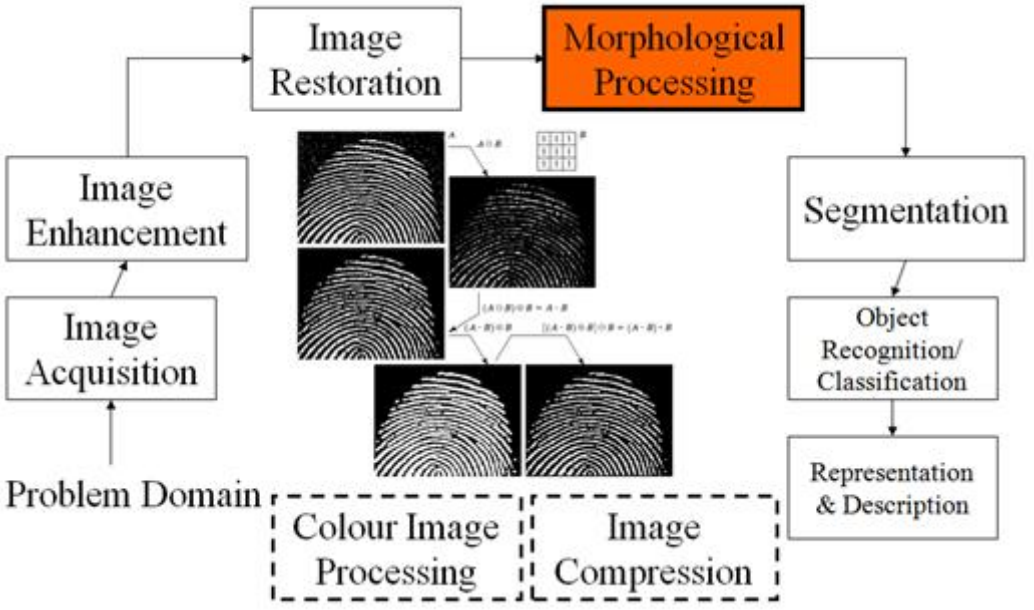
2. **Image Enhancement:** It is among the simplest and most appealing areas of digital image processing. The idea behind this is to bring out details that are obscured or simply to highlight certain features of interest in image. Image enhancement is a very subjective area of image processing and is based on human subjective preferences regarding what constitutes a "good" enhancement result.



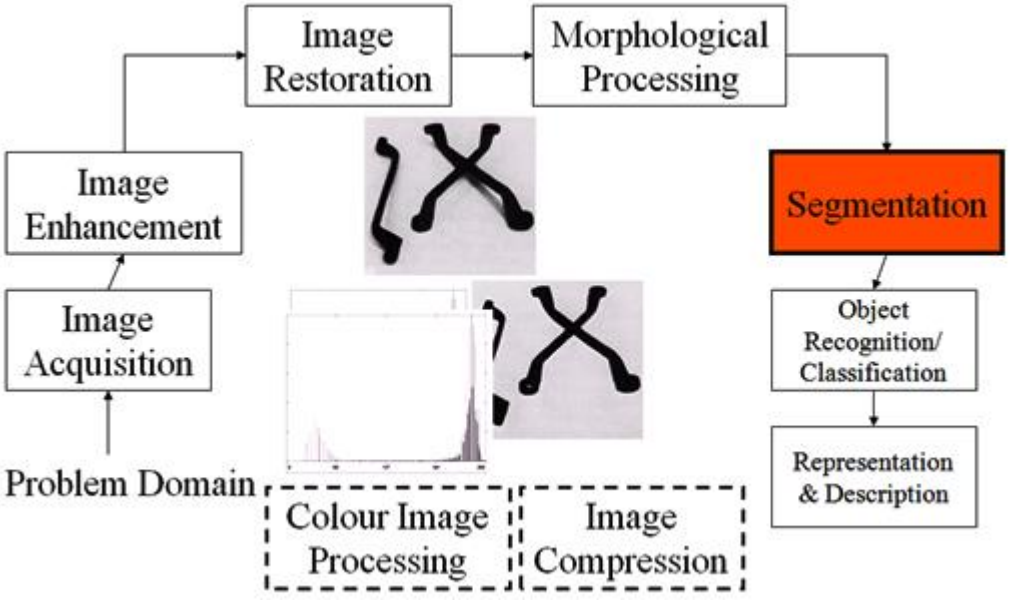
3. **Image Restoration:** It deals with improving the appearance of an image. It is an objective approach, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image processing.



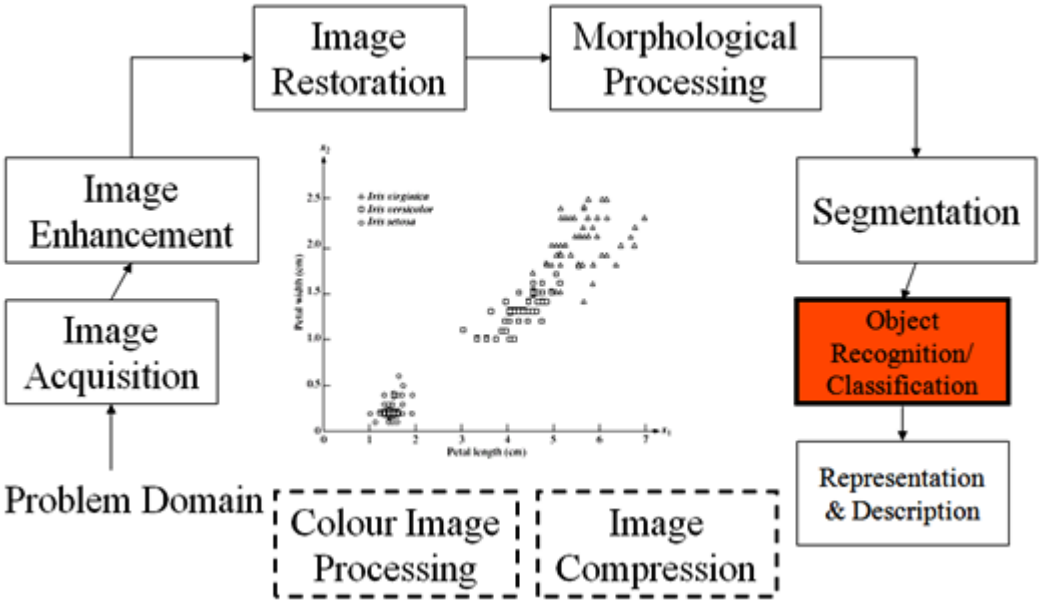
4. **Morphological processing:** It deals with tools for extracting image components that are useful in the representation and description of shape and boundary of objects. It is majorly used in automated inspection applications.



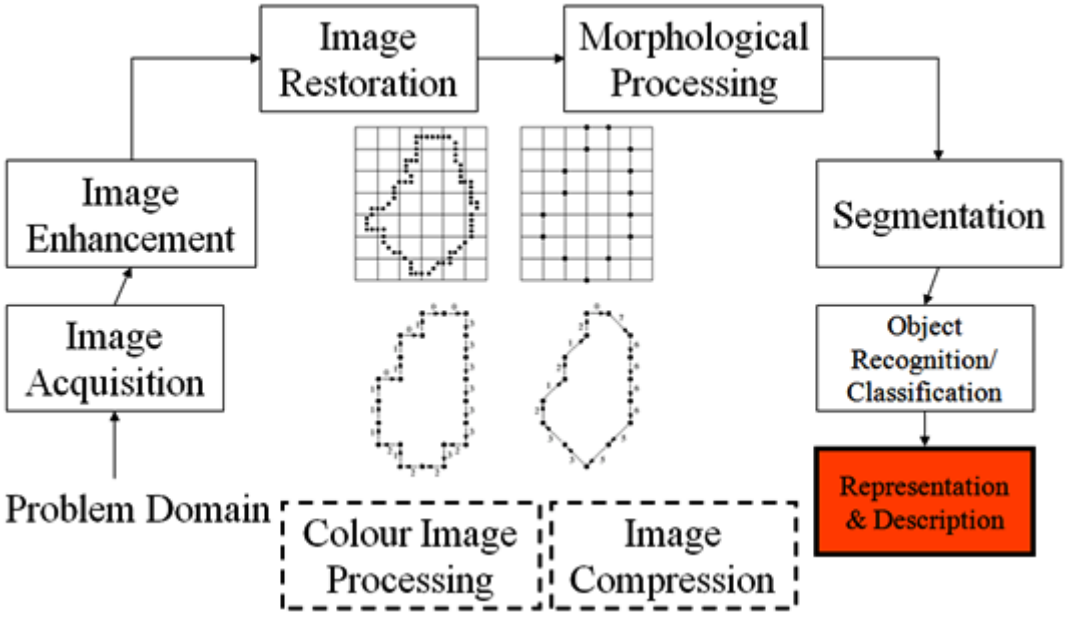
5. **Segmentation**: Image segmentation is the process of partitioning a digital image into multiple image segments, also known as image regions or image objects (sets of pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze.



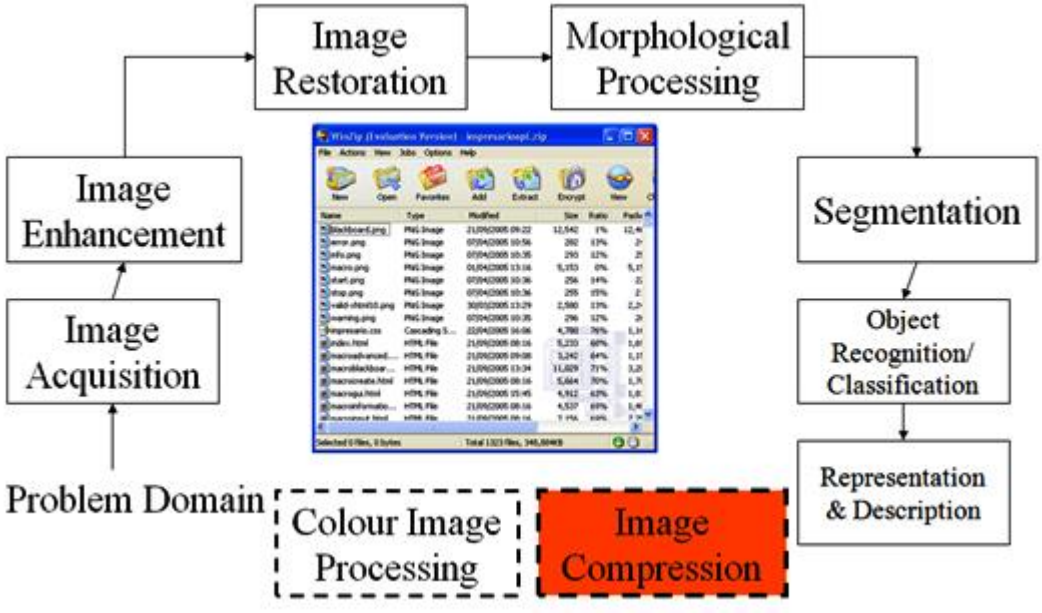
6. **Recognition:** It is the process that assigns label to an object based on its descriptors. It is the last step of image processing which uses artificial intelligence and machine learning techniques.



7. **Representation and Description:** It always follows the output of recognition step that is the labelling and describing the data based on raw pixel data.

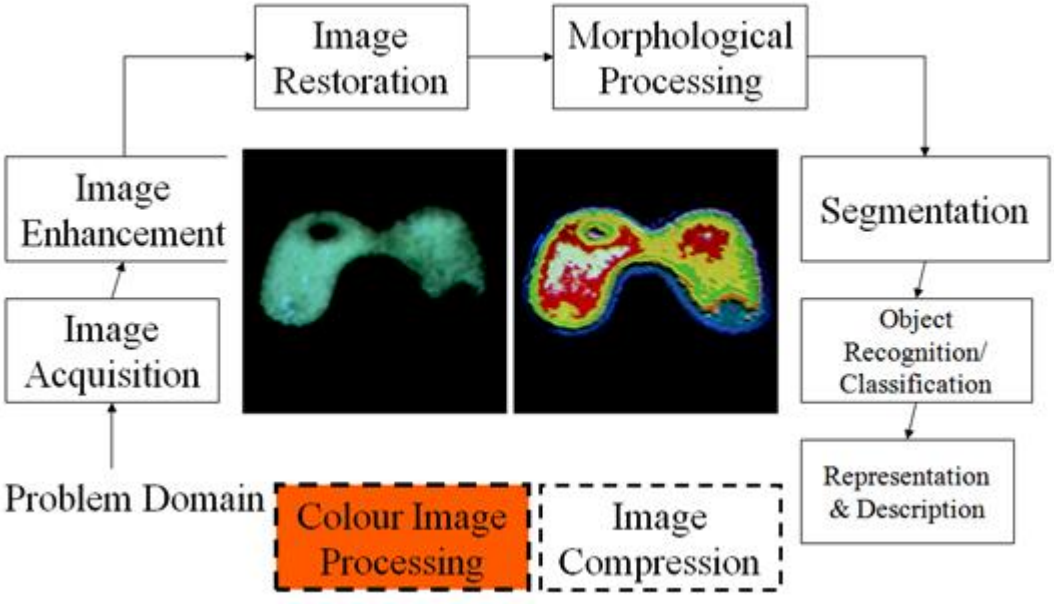


8. **Compression:** It deals with techniques reducing the storage required to save an image, or the bandwidth required to transmit it over the network. It follows either a lossy or lossless approach.



9. **Color image processing:** It is an area that is been gaining importance because of the

use of digital images over the internet and in decision making. Color image processing deals with basically color models and their implementation in image processing applications.



5. APPLICATIONS

Almost in every field, digital image processing puts a live effect on things and is growing with time to time and with new technologies.

1) Image sharpening and restoration: It refers to the process in which we can modify the look and feel of an image. It basically manipulates the images and achieves the desired output. It includes conversion, sharpening, blurring, detecting edges, retrieval, and recognition of images.

2) Medical Field: There are several applications under medical field which depends on the functioning of digital image processing.

- Gamma-ray imaging
- PET scan
- X-Ray Imaging
- Medical CT scan
- UV imaging

3) Robot vision: There are several robotic machines which work on the digital image processing. Through image processing techniques, robot finds their ways and can perform

many operations the way human does. For example, hurdle detection root, surgery and line follower robot.

4) Pattern recognition: It involves the study of image processing combined with artificial intelligence and machine learning techniques for computer-aided diagnosis, handwriting recognition and images recognition. Now a days, image processing is used for pattern recognition.

5) Video processing: It is also one of the applications of digital image processing. A collection of frames or pictures are arranged in such a way that it makes the fast movement of pictures. It involves frame rate conversion, motion detection, reduction of noise and colour space conversion etc.

9. IMAGE PROCESSING WITH PYTHON AND OPEN CV

Computer Vision can be defined as a discipline that explains how to reconstruct, interrupt, and understand a 3D scene from its 2D images, in terms of the properties of the structure present in the scene. It deals with modeling and replicating human vision using computer software and hardware.

Computer Vision overlaps significantly with the following fields −

- Image Processing − It focuses on image manipulation.
- Pattern Recognition − It explains various techniques to classify patterns.
- Photogrammetry − It is concerned with obtaining accurate measurements from images.

Here, Image processing deals with image-to-image transformation. The input and output of image processing are both images whereas computer vision is the construction of explicit, meaningful descriptions of physical objects from their image. The output of computer vision is a description or an interpretation of structures in 3D scene. We will use python with OpenCV for practical purpose. OpenCV is a cross-platform library using which we can develop real-time computer vision applications. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.

Follow the steps given below to set up OpenCV library in PyCharm IDE.

1. Download Open CV from www.opencv.org. (This experiments have been carried out on OpenCV 3.4
2. Go to File>Settings>Project:ProjectName>>Select Project Interpreter option in PyCharm.
3. Type opencv-python in serach bar.
4. Install it.
5. Similarly install matplotlib and numpy (if not available)

Following are the operations that can be performed on image in Python using OpenCV

1. READING, WRITING AND DISPLAYING IMAGE

1.1 Reading Images

Use the function cv2.imread() to read an image. The image should be in the working directory or a full path of image should be given.

Second argument is a flag which specifies the way image should be read.

cv2.IMREAD_COLOR : Loads a color image. Any transparency of image will be neglected. It is the default flag.

cv2.IMREAD_GRAYSCALE : Loads image in grayscale mode

cv2.IMREAD_UNCHANGED : Loads image as such including alpha channel

Instead of these three flags, you can simply pass integers 1, 0 or -1 respectively.

#Python code

import numpy as np

import cv2

 # Load an color image in grayscale

img = cv2.imread('xyz.jpg',0)

Even if the image path is wrong, it won't throw any error, but print img will display None

1.2 Display an Image

Use the function cv2.imshow() to display an image in a window. The window automatically fits to the image size. First argument is a window name which is a string. second argument is our image. You can create as many windows as you wish, but with different window names.

#Python code

cv2.imshow('image',img)

cv2.waitKey(0)

cv2.destroyAllWindows()

cv2.waitKey() is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If you press any key in that time, the program continues. If 0 is passed, it waits indefinitely for a key stroke. It can also be set to detect specific key strokes like, if key a is pressed etc.

cv2.destroyAllWindows() simply destroys all the windows we created. If you want to destroy any specific window, use the function cv2.destroyWindow() where you pass the exact window name as the argument.

1.3 Write Image

Use the function cv2.imwrite() to save an image.

First argument is the file name, second argument is the image you want to save.

#Python code

cv2.imwrite('xyz.png',img)

This will save the image in PNG format in the working directory.

Using Matplotlib

Matplotlib is a plotting library for Python which gives you wide variety of plotting methods. You will see them in coming articles. Here, you will learn how to display image with Matplotlib. You can zoom images, save it etc using Matplotlib.

#Python Code

import numpy as np

import cv2

from matplotlib import pyplot as plt

img = cv2.imread('xyz.jpg',0)

plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')

plt.xticks([]), plt.yticks([]) # to hide tick values on X and Y axis

plt.show()

2. BASIC OPERATIONS ON IMAGES

Following are the basic operations carried out on an image.

1. Access pixel values and modify them
2. Access image properties
3. Setting Region of Interest (ROI)
4. Splitting and Merging image

These operations require numpy but to improve performance in OpenCV, numpy is mandatory.

2.1 Accessing and Modifying pixel values

#Python code

 # Loading the image

Python code

import cv2

import numpy as np

img = cv2.imread('XYZ.jpg')

You can access a pixel value by its row and column coordinates. For BGR image, it returns an array of Blue, Green, Red values. For grayscale image, just corresponding intensity is returned.

#Python code

#Accessing pixel values

px = img[100,100]

print px

OUTPUT: [157 166 200]

 # accessing only blue pixel

blue = img[100,100,0]

print blue

OUTPUT: 157

 # Modifying pixel values

You can modify the pixel values the same way.

img[100,100] = [255,255,255]

print img[100,100]

OUTPUT: [255 255 255]

For individual pixel access, Numpy array methods, array.item() and array.itemset() is considered to be better. But it always returns a scalar. So if you want to access all B,G,R values, you need to call array.item() separately for all.

#Python code

 # accessing RED value

img.item(10,10,2)

OUTPUT: 59

 # modifying RED value

img.itemset((10,10,2),100)

img.item(10,10,2)

OUTPUT: 100

2.2 Accessing Image Properties

Image properties include number of rows, columns and channels, type of image data, number of pixels etc. Shape of image is accessed by img.shape. It returns a tuple of number of rows, columns and channels (if image is color):

#Python code

print img.shape

OUTPUT: (342, 548, 3)

Total number of pixels is accessed by img.size:

#Python code

print img.size

OUTPUT: 562248

#Python code

#Image datatype is obtained by img.dtype:

print img.dtype

OUTPUT: uint8

2.3 Setting Image ROI

Sometimes, you will have to play with certain region of images. For example, eye detection in images, first perform face detection over the image until the face is found, then search within the face region for eyes. This approach improves accuracy (because eyes are always on faces and performance (because we search for a small area).

#Python code

#ROI is again obtained using Numpy indexing.

specs = image[280:340, 200:500]

image[0:60, 0:300] = specs

cv2.imshow('roi',image)

2.4 Splitting and Merging Image Channels

The B,G,R channels of an image can be split into their individual planes when needed. Then, the individual channels can be merged back together to form a BGR image again. This can be performed by:

#Python code

b = image[:,:,0]

cv2.imshow('blue',b)

b,g,r = cv2.split(image) image = cv2.merge((b,g,r))

Suppose, you want to make all the red pixels to zero, you need not split like this and put it equal to zero. You can simply use Numpy indexing which is faster. cv2.split() is a costly operation (in terms of time), so only use it if necessary. Numpy indexing is much more efficient and should be used if possible.

If you want to create a border around the image, something like a photo frame, you can use cv2.copyMakeBorder() function. But it has more applications for convolution operation, zero padding etc. This function takes following arguments:

1. src - input image
2. top, bottom, left, right - border width in number of pixels in corresponding directions
3. borderType - Flag defining what kind of border to be added. It can be following types:
    a. cv2.BORDER_CONSTANT - Adds a constant
    b. colored border. The value should be given as next argument.
    c. cv2.BORDER_REFLECT - Border will be mirror reflection of the border elements, like this : fedcba|abcdefgh|hgfedcb
    d. cv2.BORDER_REFLECT_101 or cv2.BORDER_DEFAULT - Same as above, but with a slight change, like this : gfedcb|abcdefgh|gfedcba
    e. cv2.BORDER_REPLICATE - Last element is replicated throughout, like this: aaaaaa|abcdefgh|hhhhhhh

f. cv2.BORDER_WRAP - Random, it will look like this : cdefgh|abcdefgh|abcdefg

4. value - Color of border if border type is cv2.BORDER_CONSTANT

3. ARITHMETIC OPERATIONS ON IMAGES

Following arithmetic operations can be performed on images.

1. Image Addition
2. Image Blending
3. Bitwise Operations

3.1 Image Addition

You can add two images by OpenCV function, cv2.add() or simply by numpy operation,

res = img1 + img2.

Both images should be of same depth and type, or second image can just be a scalar value. There is a difference between OpenCV addition and Numpy addition. OpenCV addition is a saturated operation while Numpy addition is a modulo operation.

x = np.uint8([250])

y = np.uint8([10])

print cv2.add(x,y) # 250+10 = 260 => 255 [[255]]

print x+y # 250+10 = 260 % 256 = 4 [4]

It will be more visible when you add two images. OpenCV function will provide a better result. So always better stick to OpenCV functions.

3.2 Image Blending

This is also image addition, but different weights are given to images so that it gives a feeling of blending or transparency. Here, I took two images to blend them together. First image is given a weight of 0.7 and second image is given 0.3.

cv2.addWeighted() applies following equation on the image.

#Python code

img1 = cv2.imread('1.png')

img2 = cv2.imread('Taj.jpg')

dst = cv2.addWeighted(img1,0.7,img2,0.3,0)

cv2.imshow('dst',dst)

cv2.waitKey(0)

cv2.destroyAllWindows()

3.3 Bitwise Operations

This includes bitwise AND, OR, NOT and XOR operations. They will be highly useful while extracting any part of the image, defining and working with non-rectangular ROI etc. Here, we will see an example on how to change a particular region of an image. Bitwise Operations are required for the following purpose.

1. If we to put some logo above an image.
2. If we add two images, it will change color.
3. If we blend it, we get transparent effect.
4. Especially, when the image is not in rectangular shape.
5. For this, we have to go for bitwise operation.

10. REFERENCES

Following are the references that have been used to create this book. I am thankful to these contributors for providing these resources.

1. Computer Graphics C Version by Donald Hearn & M Pauline Baker II Edition
2. https://lectureloops.com/computer-graphics/
3. https://www.tutorialandexample.com/computer-graphics-tutorial
4. https://www.includehelp.com/computer-graphics/
5. https://www.geeksforgeeks.org/computer-graphics-2/
6. https://www.gatevidyalay.com/computer-graphics/
7. https://www.javatpoint.com/computer-graphics-tutorial
8. https://www.tutorialspoint.com/computer_graphics/index.htm
9. http://www.gpcet.ac.in/wp-content/uploads/2018/08/DIP-UNIT-1.pdf
10. https://kanchiuniv.ac.in/coursematerials/Digital%20image%20processing%20-Vijaya%20Raghavan.pdf
11. http://library.iiests.ac.in:30000/dqpas/pdf/Contents_Image_Processing_Ajoy_Kumar_Ray.pdf
12. https://www.drkmm.com/resources/INTRODUCTION_TO_IMAGE_PROCESSING_29aug06.pdf
13. http://library.iiests.ac.in:30000/dqpas/pdf/Contents_Image_Processing_Ajoy_Kumar_Ray.pdf
14. https://gacbe.ac.in/pdf/ematerial/20MCA24C-U1.pdf
15. https://getsetcg.blogspot.com/