UNIT 1

INTRODUCTION TO COMPUTER GRAPHICS

1.1) INTRODUCTION

Computer graphics is an art of drawing pictures, lines, charts, etc using computers with the help of programming. Computer graphics is made up of number of pixels. Pixel is the smallest graphical picture or unit represented on the computer screen. Basically there are two types of computer graphics namely.

Interactive Computer Graphics: Interactive Computer Graphics involves a two way communication between computer and user. Here the observer is given some control over the image by providing him with an input device for example the video game controller of the ping pong game. This helps him to signal his request to the computer.

The computer on receiving signals from the input device can modify the displayed picture appropriately. To the user it appears that the picture is changing instantaneously in response to his commands. He can give a series of commands, each one generating a graphical response from the computer. In this way he maintains a conversation, or dialogue, with the computer.

Interactive computer graphics affects our lives in a number of indirect ways. For example, it helps to train the pilots of our airplanes. We can create a flight simulator which may help the pilots to get trained not in a real aircraft but on the grounds at the control of the flight simulator. The flight simulator is a mock up of an aircraft flight deck, containing all the usual controls and surrounded by screens on which we have the projected computer generated views of the terrain visible on take off and landing.

Flight simulators have many advantages over the real aircrafts for training purposes, including fuel savings, safety, and the ability to familiarize the trainee with a large number of the world's airports.

Non Interactive Computer Graphics: In non interactive computer graphics otherwise known as passive computer graphics. it is the computer graphics in which user does not have any kind of control over the image. Image is merely the product of static stored program and will work according to the instructions given in the program linearly. The image is totally under the control of program instructions not under the user. Example: screen savers.

Applications of Computer Graphics

The following are also considered graphics applications.

Paint programs: Allow you to create rough freehand drawings. The images are stored as bit maps and can easily be edited. It is a graphics program that enables you to draw pictures on the display screen which is represented as bit maps (bit-mapped graphics). In contrast, draw programs use vector graphics (object-oriented images), which scale better.

Most paint programs provide the tools shown below in the form of icons. By selecting an icon, you can perform functions associated with the tool.In addition to these tools, paint programs also provide easy ways to draw common shapes such as straight lines, rectangles, circles, and ovals.

Sophisticated paint applications are often called image editing programs. These applications support many of the features of draw programs, such as the ability to work with objects. Each object, however, is represented as a bit map rather than as a vector image.

Illustration/design programs: Supports more advanced features than paint programs, particularly for drawing curved lines. The images are usually stored in vector-based formats. Illustration/design programs are often called draw programs. Presentation graphics software: Lets you create bar charts, pie charts, graphics, and other types of images for slide shows and reports. The charts can be based on data imported from spreadsheet applications.

A type of business software that enables users to create highly stylized images for slide shows and reports. The software includes functions for creating various types of charts and graphs and for inserting text in a variety of fonts. Most systems enable you to import data from a spreadsheet application to create the charts and graphs. Presentation graphics is often called business graphics.

Animation software: Enables you to chain and sequence a series of images to simulate movement. Each image is like a frame in a movie. It can be defined as a simulation of movement created by displaying a series of pictures, or frames. A cartoon on television is one example of animation. Animation on computers is one of the chief ingredients of multimedia presentations. There are many software applications that enable you to create animations that you can display on a computer monitor.

There is a difference between animation and video. Whereas video takes continuous motion and breaks it up into discrete frames, animation starts with independent pictures and puts them together to form the illusion of continuous motion.

CAD software: Enables architects and engineers to draft designs. It is the acronym for computer-aided design. A CAD system is a combination of hardware and software that enables engineers and architects to design everything from furniture to airplanes. In addition to the software, CAD systems require a high-quality graphics monitor; a mouse, light pen, or digitizing tablet for drawing; and a special printer or plotter for printing design specifications.

CAD systems allow an engineer to view a design from any angle with the push of a button and to zoom in or out for close-ups and long-distance views. In addition, the computer keeps track of design dependencies so that when the engineer changes one value, all other values that depend on it are automatically changed accordingly. Until the mid 1980s, all CAD systems were specially constructed computers. Now, you can buy CAD software that runs on general-purpose workstations and personal computers.

Desktop publishing: Provides a full set of word-processing features as well as fine control over placement of text and graphics, so that you can create newsletters, advertisements, books, and other types of documents. It means by using a personal computer or workstation high-quality printed documents can be produced. A desktop publishing system allows you to use different typefaces, specify various margins and justifications, and embed illustrations and graphs directly into the text. The most powerful desktop publishing systems enable you to

create illustrations; while less powerful systems let you insert illustrations created by other programs.

As word-processing programs become more and more powerful, the line separating such programs from desktop publishing systems is becoming blurred. In general, though, desktop publishing applications give you more control over typographical characteristics, such as kerning, and provide more support for full-color output.

A particularly important feature of desktop publishing systems is that they enable you to see on the display screen exactly how the document will appear when printed. Systems that support this feature are called WYSIWYGs (what you see is what you get). Until recently, hardware costs made desktop publishing systems impractical for most uses. But as the prices of personal computers and printers have fallen, desktop publishing systems have become increasingly popular for producing newsletters, brochures, books, and other documents that formerly required a typesetter.

Once you have produced a document with a desktop publishing system, you can output it directly to a printer or you can produce a PostScript file which you can then take to a service bureau. The service bureau has special machines that convert the PostScript file to film, which can then be used to make plates for offset printing. Offset printing produces higher-quality documents, especially if color is used, but is generally more expensive than laser printing.

In general, applications that support graphics require a powerful CPU and a large amount of memory. Many graphics applications—for example, computer animation systems—require more computing power than is available on personal computers and will run only on powerful workstations or specially designed graphics computers. This is true of all three-dimensional computer graphics applications.

In addition to the CPU and memory, graphics software requires a graphics monitor and support for one of the many graphics standards. Most PC programs, for instance, require VGA graphics. If your computer does not have built-in support for a specific graphics system, you can insert a video adapter card.

The quality of most graphics devices is determined by their resolution—how many pixels per square inch they can represent—and their color capabilities.

Vector Scan Display	Raster Scan Display
1. In vector scan display the beam is moved between the end points of the graphics primitives.	1. In raster scan display the beam is moved all over the screen one scan line at a time, from top bottom and then back to top,
2. Vector display flickers when the number of primitives in the buffer becomes too large.	2. In raster display, the refresh process is independent of the complexity of the image.
3. Scan conversion is not required.	3. Graphics primitives are specified in terms of their endpoints and must be scan converted into their corresponding pixels in the frame buffer.
4. Scan conversion hardware is not required.	4. Because each primitive must be scan-converted,real time dynamics is for more computational and requires separate scan conversion hardware.
5. Vector display draws a continuous and smooth lines.	5. Raster display can display mathematically smooth lines, polygons, and boundaries of curved primitives only by approximating them with pixels on the raster grid.
6. Cost is more.	6. Cost is low.
7. Vector display only draws lines and characters.	7. Raster display has ability to display areas filled with solid colours or patterns.

1.2) ELEMENTS OF THE COMPUTER GRAPHICS

Two types of graphic display are supported on the Commodore 128 (or C-128, for short). These are known as the low-resolution (or character) images and the high-resolution (or bit-mapped) images. On the C-128 the former come in two formats: 40 columns and 80 columns. Each of these formats is controlled by a separate microprocessor. When the 80-column format is selected, especially useful for word-processing, only the keyboard characters can be displayed on the screen. On the other hand, the microchip controlling the 40-column format supports, in addition to all the keyboard characters (low-resolution mode), also the other, the high-resolution mode. In this book we concentrate entirely on high-resolution graphics and we shall therefore disregard the 80-column format completely. Consequently, in this text, any reference to the low-resolution screen implies the 40-column format.

Display file

The Commodore 128, like other microcomputers, reserves an area in memory designated for the graphic display. This part of the memory contains all the information needed to produce an image. In this book, the area allocated to graphics is referred to as the display file. In such a file the image is stored as a set of binary numbers, which, in turn, represent the picture elements. These elements are the smallest units of the image that we can manipulate and are known as pixels. In the high-resolution mode each binary digit (or bit) in a display file represents a single pixel. In this case the reverse statement is also valid, that is, each pixel has a single counterpart in the display file. Later on we shall see that another mode is available where this one-to-one correspondence is no longer valid. At first, it may sound bewildering that a picture on the screen is represented by a series of numbers. How is such a coding achieved? First of all, the display area of the screen is divided into a large number of min cells. Think of it as a grid with 320 divisions in the horizontal direction and 200 divisions in the vertical direction. Consequently, the usable area of the screen is divided into 64 000 (320 x 200) cells. These are the smallest areas that can be individually manipulated and are, in fact, identified with pixels.

The size of the pixels is determined by the computer as well as by the display tube. The numbers just mentioned apply to the C-128. The larger the number of pixels, the higher the resolution. In other words, the finer the grid is, the more detailed and smoother our display will be. The size of the pixels is determined at the manufacturing stage and we cannot alter it (but see the discussion below of the multicolor mode). As the screen is now divided into 64 000 pixels, the display file, containing one bit per pixel, holds 64 000 bits. Put otherwise, the display area of the screen in the high-resolution mode requires 8000 bytes (= 64 000 bits) of memory. The coding of the display file now becomes possible. The bits, being digits of a binary system, can be either 0 or 1. If a bit in the display file is equal to 0 (off), then the corresponding location on the screen has the colour of the background, that is, the corresponding pixel is transparent and no visible change takes place on the screen. On the other hand, if a bit has a

value of 1 (on), then a dot of the specified color appears in the corresponding location on the screen. This conversion between individual bits and dots is performed by the part of the C-128 dedicated to the video display. The essence of picture production is as follows. Each byte of the display file is read and converted into the appropriate video signal. This, in turn, is fed into the video monitor (commonly a TV set) where the signal is transformed into on and off dots, depending on the value of the individual bits. The image is formed with the help of the electron beam that scans the phosphor-covered display area of the TV tube. This beam travels from top to bottom and from left to right. That is, the scan begins at the top left corner, traverses to the right, then returns to the left and begins at the next line, and so on till the end of the display. When the electron beam reaches the bottom right-hand corner, it returns to the top left-hand corner and the process is repeated. The whole operation is very fast and the screen is fully scanned 50 times each second. The display area consists of 200 horizontal lines (traced by the electron beam), also known as rasters. Each raster contains 320 pixels. The task of producing a display file from a given picture consists of two steps. First, the continuous image, such as one would normally draw by hand, is replaced by an image constructed from distinct pixels, the so-called discrete image. Then, each pixel is coded and entered into the appropriate position in the display file. In this way a picture is separated into its elements, leading to the creation of the relevant binary display file.

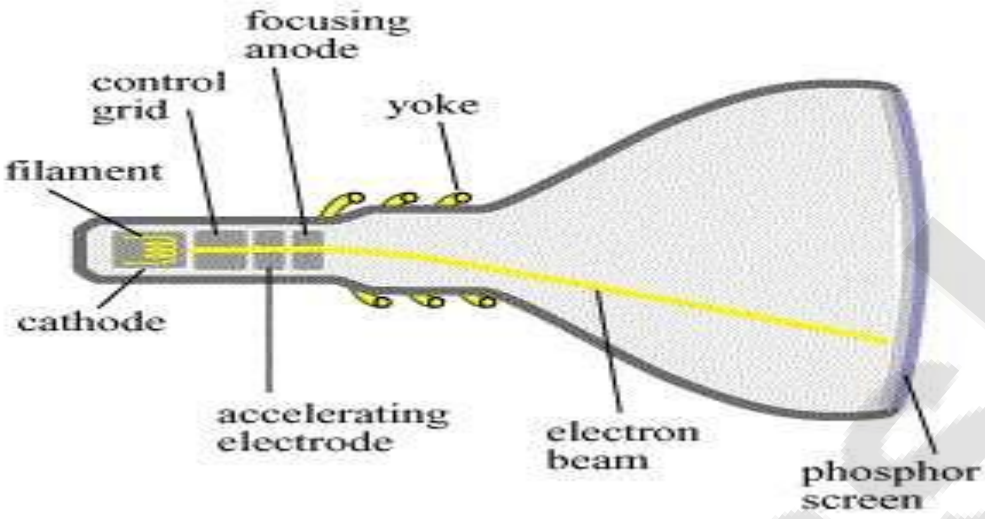
1.3) GRAPHICS DISPLAY SYSTEM

Cathode Ray Tube (CRT/Monitor)

One of the basic and commonly used display devices is Cathode Ray Tube (CRT). A cathode ray tube is based on the simple concept that an electronic beam, when hits a phosphorescent surface, produces a beam of light (momentarily - though we later describe surfaces that produce light intensities lashing over a period of time).

Further, the beam of light itself can be focused to any point on the screen by using suitable electronic / magnetic fields. The direction and intensity of the fields will allow one to determine the extent of the defection of the beam. Further these electronic / magnetic fields can be easily manipulated by using suitable electric fields with this background. In following section we describe the structure and working of the simple CRT.

Simple CRT makes use of a conical glass tube. At the narrow end of the glass tube an electronic gun is kept. This gun generates electrons that will be made to pass through the magnetic system called yoke. This magnetic system is used for making the electronic beam to fall throughout the broad surface of the glass tube. The broad surface of the glass tube contains a single coat of high quality phosphorus. This reflects the electronic beam makes it to fall on the computer screen.



A pair of focusing grids - one horizontal and another vertical does the actual focusing of the electronic beam on to the screen. Electronic or magnetic fields operate these grids. Depending on the direction (positive or negative) and the intensity of the fields applied to them, the beam is deflected horizontally (or vertically) and thus, by using a suitable combination of these focusing grids; the beam can be focused to any point on the screen. So, we now have a mechanism wherein any point on the screen can be illuminated (or made dark by simply switching off the beam). Hence, from a graphics point of view, any picture can be traced on the screen by the electron beam by suitably and continuously manipulating the focusing grids and we get to see the picture on the screen "A basic graphic picture" of course, since the picture produced vanishes once the beam is removed, to give the effect to continuity, we have to keep the beam retracing the picture continuously.

Shadow Mask CRT (Cathode Ray Tube)

In Shadow Mask CRT tiny holes in a metal plate separate the colored phosphors in the layer behind the front glass of the screen. The holes are placed in a manner ensuring that electrons from each of the tube's three cathode guns reach only the appropriately-colored phosphors on the display. All three beams pass through the same holes in the mask, but the angle of approach is different for each gun.

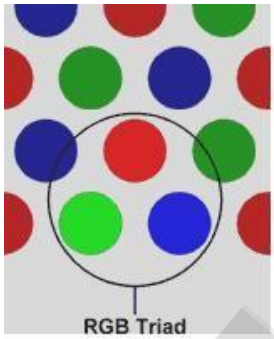
The spacing of the holes, the spacing of the phosphors, and the placement of the guns is arranged so that for example the blue gun only has an unobstructed path to blue phosphors. The red, green, and blue phosphors for each pixel are generally arranged in a triangular shape (sometimes called a "triad"). All early color televisions and the majority of computer monitors, past and present, use shadow mask technology.

Traditionally, shadow masks have been made of materials which temperature variations cause to expand and contract to the point of affecting performance. The energy the shadow mask absorbs from the electron gun in normal operation causes it to heat up and expand, which leads to blurred or discolored (see doming) images. The invar shadow mask is composed of the nickel-iron alloy invar.

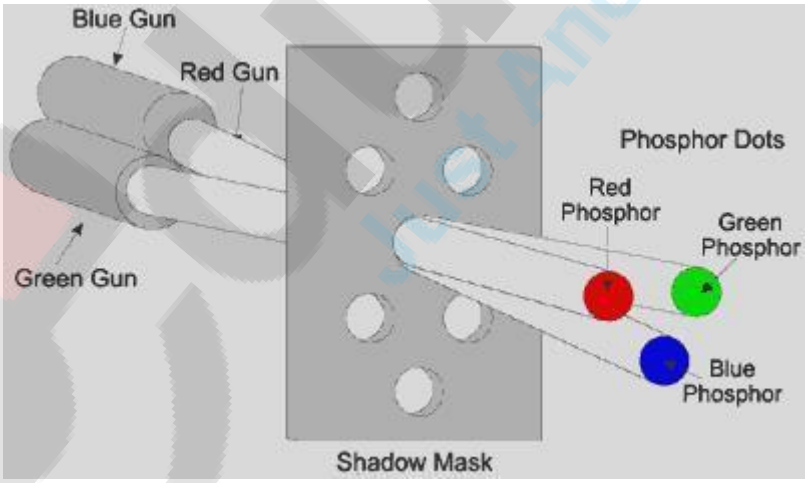
Therefore it expands and contracts much less than other materials in response to temperature changes. This property allows displays made with this technology to provide a

clearer, more accurate picture. It also reduces the amount of long-term stress and damage to the shadow mask that can result from repeated expand/contract cycles, thus increasing the display's life expectancy.

In other words, In Shadow Mask CRT, before the stream of electrons produced by the CRT's cathode reach the phosphor coated faceplate, it encounters the shadow mask, a sheet of metal etched with a pattern of holes. The mask is positioned in the glass funnel of the CRT during manufacture and the phosphor is coated onto the screen so that electrons coming from the red, green and blue gun positions only land on the appropriate phosphor.



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



An alternative to the shadow mask which is less prone to distortion, the aperture grille , was included as part of the design of Trinitron CRTs by Sony in 1968 and Mitsubishi in its Diamondtron products in the early 1990s.

Liquid Crystal Display

LCD stands for liquid crystal display. Your *digital* watch uses an LCD to show you the time, and most portable computers use an LCD to display the screen. There is actually a liquid

compound, liquid crystals, sandwiched between two grids of electrodes. The electrodes can selectively turn on the different cells or *pixels* in the grid to create the image you see.

An LCD consists of a layer of gooey material-the liquid crystals themselves-between two polarizing filters. These filters are sheets of plastic that let through only those light waves traveling parallel to a particular plane. Between the filters and the liquid crystal layer runs a thin grid of transparent electrodes.

The two polarizing filters are arranged so that their polarizing planes are at right angles. That setup would block light from passing through except for the fact that the liquid crystal molecules are "twisted." They pivot the light coming through the first filter, aligning the light with the polarizing plane of the second filter. Since the light makes it all the way through both filters, the screen looks light in color. However, the liquid crystal molecules that are controlled by a particular electrode become untwisted when a current is applied. Light no longer passes through the second filter, and you see a black or colored dot on the screen. Most LCDs are passive matrix designs, in which each dot, or pixel, on the screen shares electrodes with other dots. *Active matrix* designs, which produce much brighter, more colorful images, have a separate transistor for each pixel, which allows greater control over the current for that pixel.

In "supertwist" LCDS, the liquid crystal molecules have a more pronounced twist than in the ordinary screens, improving contrast. The chemist's term "nematic" refers to the molecular structure of the crystals-all LCDs use nematic crystals, so this term is used in ads just to impress you.

Although you can read an LCD screen in room light, the contrast is mediocre at best. Today, the LCD screens on most computers are illuminated by backlighting or edge lighting (fluorescent-type lights mounted behind the screen or along either side).

UNIT 2

OUTPUT PRIMITIVES AND IT'S ALORITHMS

2.1) Points and Lines

Point plotting is done by converting a single coordinate position furnished by an application program into appropriate operations for the output device in use.

Line drawing was done by calculating intermediate positions along the line path between two specified endpoint positions.

The output device then directed to fill in those positions between the endpoints with some color.

For some device such as a pen plotter or random scan display, a straight line can draw smoothly from one endpoint to other.

Digital devices display a straight line segment by plotting discrete points between the two endpoints.

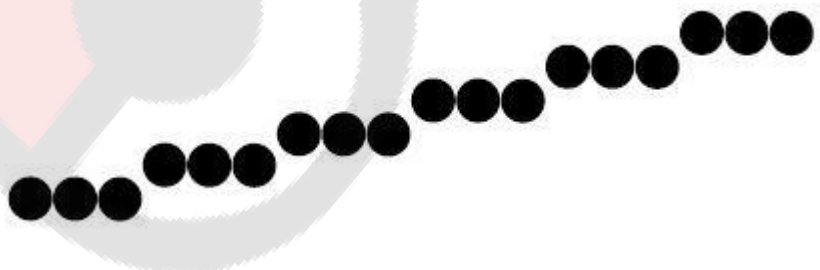
Discrete coordinate positions along the line path calculated from the equation of the line.

For a raster video display, the line intensity loaded in the frame buffer at the corresponding pixel positions.

Reading from the frame buffer, the video controller then plots the screen pixels.

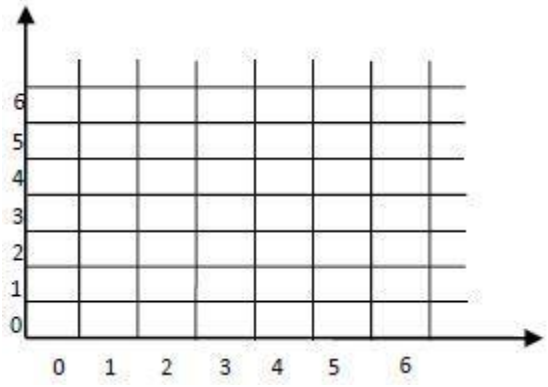
Screen locations referenced with integer values, so plotted positions may only approximate actual line positions between two specified endpoints.

For example line position of (12.36,23.87) would converted to pixel position (12,24).



- This rounding of coordinate values to integers causes lines to display with a stair step appearance ("the jaggies"), as represented in fig.

- The stair step shape is noticeable in a low-resolution system, and we can improve their appearance somewhat by displaying them on a high-resolution system.
- More effective techniques for smoothing raster lines based on adjusting pixel intensities along the line paths.
- For raster graphics device-level algorithms discuss here, object positions are specified directly in integer device coordinates.



- Pixel position will reference according to the scan-line number and column number which illustrated by the following figure.
- To load the specified color into the frame buffer at a particular position. We will assume we have available low-level procedure of the form $setpixe(x, y)$.
- Similarly, for retrieving the current frame buffer intensity we assume to have procedure $getpixe(x, y)$.

2.2) LINE DRAWING ALGORITHMS

2.2.1)Digital Differential Analyzer (DDA)

Digital Differential Analyzer (DDA) algorithm is the simple line generation algorithm which is explained step by step here.

Step 1 − Get the input of two end points (X0,Y0)(X0,Y0) and (X1,Y1)(X1,Y1).

Step 2 − Calculate the difference between two end points.

```
dx = X1 - X0
dy = Y1 - Y0
```

Step 3 − Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.

```
if (absolute(dx) > absolute(dy))
  Steps = absolute(dx);
else
```

```
Steps = absolute(dy);
```

Step 4 − Calculate the increment in x coordinate and y coordinate.

```
Xincrement = dx / (float) steps;
Yincrement = dy / (float) steps;
```

Step 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++)
{
  x = x + Xincrement;
  y = y + Yincrement;
  putpixel(Round(x), Round(y));
}
```

DDA (Digital Differential Analyzer) Line Drawing Algorithm:

It is a mechanical device for integrating differential equations by simultaneously incrementing x and y in small steps proportional to DX and DY.
The line end points are (xa,ya) and (xb,yb).

Step 1: Compute DX and DY
DX=xb-xa
DY=yb-ya

Step 2: Calculate Steps
If abs(DX) > abs(DY) Then
Steps=abs(DX)
Else
Steps=abs(DY)
End if

Step 3: Calculate Xincrement and Yincrement
Xincr=DX/Steps
Yinct=DY/Steps

Step 4: Set Initial Point to x,y
X=xa
Y=ya
I=1

Step 5: Repeat while I<=Steps

SetPixel(x,y)
X=x+xincr
Y=y+yincr
I=I+1
End While

2.2.2) Bresenham Algorithm

- Bresenham Algorithm was developed by J.E.Bresenham in 1962 and it is much accurate and much more efficient than DDA.
- It scans the coordinates but instead of rounding them off it takes the incremental value in account by adding or subtracting and therefore can be used for drawing circle and curves.
- Therefore if a line is to be drawn between two points x and y then next coordinates will be(xa+1, ya) and (xa+1, ya+1) where a is the incremental value of the next coordinates and difference between these two will be calculated by subtracting or adding the equations formed by them.

Bresenham's Line-Drawing Algorithm
An accurate and efficient line-generating algorithm , developed by Bresenham , scan converts lines using only incremental integer calculations that can be adapted to display circles and other curves.
Pixel positions along a line path are determined by sampling at unit x intervals. Starting from the left end point (X_0,Y_0) of a given line,we step to each successive column (x position) and plot the pixel whose scan-line y value is closed to the line path.
Assuming we have determined that the pixel at (X_k,Y_k) is to be displayed, we next need to decide which pixel to plot in column X_{k+1}. Our choices are the pixels at positions (X_{k+1},Y_k) and (X_{k+1},Y_{k+1}).
If pixel Y_k is closer to the line path than the pixel at Y_{k+1} , then decision parameter P_k is negative. In that case , we plot the lower pixel; otherwise , we plot the upper pixel.
The term $Y_{k+1} - Y_k$ is either 0 or 1, depending on the sign of parameter P_k.
The first parameter P_0 is evaluated at the starting pixel position (X_0,Y_0) and with M evaluated as DY/DX . **P_0=2DY-DX**.
The constants are **2DY** if P_k <0 and **2DY-2DX** if P_k>0 are calculated for each line to be scan converted, so the arithmetic involves only integer addition and subtraction of these two constants.

Bresenham's Line-Drawing Algorithm for m < 1 (gentle slope)

Step 1: Input the two line end points.

Step 2: Calculate constants. Obtain the starting value for the decision parameter.
DX=abs(xa-xb)
DY=abs(ya-yb)
P=**2DY-DX**

Inc1=**2DY**
Inc2=**2DY-2DX**

Step 3: Determine which point to use as start, which as end.
If (xa < xb)
X=xa
Y=ya
Xend=xb
Else
X=xb
Y=yb
Xend=xa

Step 4: At each X_k along the line, starting at k=0, perform the following test.
If $P_k<0$, the next point to plot is (X_{k+1},Y_k) and
$P_k=P_k+inc1$
Otherwise , the next point to plot is (X_{k+1},Y_{k+1}) and
$P_{k+1}=P_k+inc2$

Step 5: Plot the First Pixel.

Step 6 : Repeat step 4 and 5 , DX times.

Eduklash Just Another Way To Learn

	Digital Differential Analyzer Line Drawing Algorithm	Bresenhams Line Drawing Algorithm
Arithmetic	DDA algorithm uses **floating points** i.e. **Real Arithmetic**.	Bresenhams algorithm uses **fixed points** i.e. **Integer Arithmetic**.
Operations	DDA algorithm uses **multiplication** and **division** in its operations.	Bresenhams algorithm uses only **subtraction** and **addition** in its operations.
Speed	DDA algorithm is rather **slow**ly than Bresenhams algorithm in line drawing because it uses real arithmetic (floating-point operations).	Bresenhams algorithm is faster than DDA algorithm in line drawing because it performs only addition and subtraction in its calculation and uses only integer arithmetic so it runs significantly **faster**.
Accuracy & Efficiency	DDA algorithm is not as accurate and efficient as Bresenham algorithm.	Bresenhams algorithm is more efficient and much accurate than DDA algorithm.
Drawing	DDA algorithm can draw circles and curves but that are not as accurate as Bresenhams algorithm.	Bresenhams algorithm can draw circles and curves with much more accuracy than DDA algorithm.
Round Off	DDA algorithm round off the coordinates to integer that is nearest to the line.	Bresenhams algorithm does not **round off** but takes the incremental value in its operation.
Expensive	DDA algorithm uses an enormous number of floating-point multiplications so it is expensive.	Bresenhams algorithm is less expensive than DDA algorithm as it uses only addition and subtraction.

2.3)CIRCLE AND ELLIPSE GENERATING ALGORITHMS

2.3.1) MidPoint Circle Algorithm:

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

Along the circle section from x=0 to x=y in the first quadrant, the slope of the curve varies from 0 to –1. therefore , we can take unit steps in the positive x direction over this octant and use a decision parameter to determine which of the two possible y positions is closer to the circle path at each step. Positions in the other seven octants are then obtained by symmetry.

To apply the midpoint method, we define a circle function: $Fcircle(x,y) = x^2+y^2-r^2$

Fcircle(x,y) = 0 , if (x,y) is on the circle boundary

< 0 , if (x,y) is inside the circle boundary

> 0 , if (x,y) is outside the circle boundary.

The circle function is the decision parameter in the midpoint algorithm and we can set up incremental calculations for this function.

Increments are either **2X+1** if p is negative or **2X+1-2Y** if p is positive.

The initial decision parameter is obtained by evaluating the circle function at the start position $(X_0,Y_0)=(0,r)$: $P_0=fcircle(1,r-1/2) = 1+(r-1/2)^2 – r^2 = 5/4-r = 1-r.$

MidPoint Circle Algorithm:

Step 1: Input radius r and circle center (Xc,Yc) and obtain the first point on the circle circumference of a circle centered on the origin as $(X_0,Y_0)=(0,r)$

Step 2: Calculate the initial value of the decision parameter as $P_0=$**1-r**

Step 3: At each X_k position, starting at k=0, perform the following test.
If $P_k<0$, the next point along the circle centered on (0,0) is (X_{k+1},Y_k) and $P_{k+1}=P_k+2X_{k+1}+1$
Otherwise, the next point along the circle is (X_{k+1},Y_{k-1}) and $P_{k+1}=P_k+2X_{k+1}+1-2Y_{k+1}$

Step 4: Determine symmetry points in the other seven octants. Move each calculated pixel position (X,Y) onto the circle path centered on (Xc,Yc) and plot the coordinate values.
X=X+Xc, Y=Y+Yc

Step 5: Repeat step 3 through 4 till X<=Y.

2.3.2) Mid-Point Ellipse Algorithm:

1. Read radii rx and ry
2. Initialize starting point x=0
y=ry
3. Calculate initial value of decision parameter in region 1:
$d1=ry^2-rx^2ry + (1/4)\ rx^2$
4. Initialize dx and dy
$dx= 2\ ry^2x$
$dy= 2\ rx^2y$
5. do
{
Plot (x,y)
If (d1<0)
{
x=x+1
y=y
$dx= dx+ 2ry^2$
$d1= d1+ dx+ ry^2$
$[\ d1= d1+ 2\ ry^2x + 2ry^2 + ry^2]\backslash$
}
else
{
x=x+1
y=y-1
$dx= dx+ 2\ ry^2$
$dy=dy- 2\ rx^2$
$d1= d1+ dx-dy + ry^2$
$[\ d1= d1+ d1+ 2\ ry^2x + 2ry^2 – (2\ rx^2y -2\ rx^2) + ry^2]$

}
While (dx<dy)
6. Calculate initial value of decision parameter in region 2:
$d2= ry^2 (x+ (1/2))^2 + rx^2 (y-1)^2 – rx^2 ry^2$
7. do
{
Plot (x,y)
If (d2>0)
{
x=x
y=y-1
$dy=dy-2 rx^2$
$d2=d2-dy+ rx^2$
$[d2= d2- (2 rx^2y – 2rx^2) + rx^2]$
}
else
{
x=x+1
y=y-1
$dy=dy-2 rx^2$
$dx= dx+ ry^2$
$d2= d2+dx-dy+ rx^2$
$[d2= d2+ 2 ry^2x + 2 ry^2 – (2 rx^2y – 2rx^2) + ry^2]$
}
While (y>0)
8. Determine symmetrical points in other 3 quadrants.
9. Stop.

2.4) PARAMETRIC CUBIC CURVES

2.4.1) Bezier Curves

Bezier curve is discovered by the French engineer **Pierre Bézier**. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as −

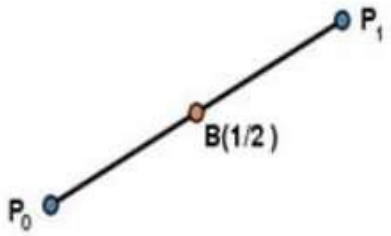
$$\sum_{k=0}^{n} P_i B_i^n(t)$$

Where pipi is the set of points and $B_i^n(t)$ represents the Bernstein polynomials which are given by −

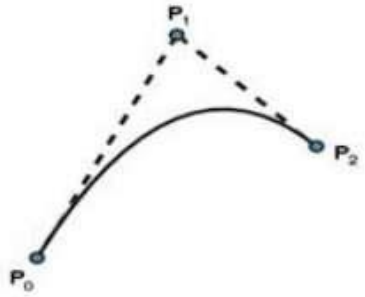
$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i$$

Where **n** is the polynomial degree, **i** is the index, and **t** is the variable.

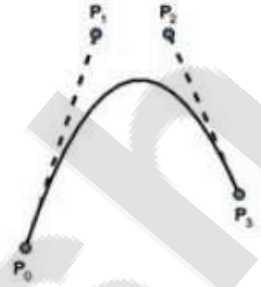
The simplest Bézier curve is the straight line from the point P0P0 to P1P1. A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is determined by four control points.



Simple Bezier Curve



Quadratic Bazier Curve



Cubic Bazier Curve

Properties of Bezier Curves

Bezier curves have the following properties −

- They generally follow the shape of the control polygon, which consists of the segments joining the control points.
- They always pass through the first and last control points.
- They are contained in the convex hull of their defining control points.
- The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.
- A Bezier curve generally follows the shape of the defining polygon.
- The direction of the tangent vector at the end points is same as that of the vector determined by first and last segments.
- The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
- No straight line intersects a Bezier curve more times than it intersects its control polygon.
- They are invariant under an affine transformation.
- Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.
- A given Bezier curve can be subdivided at a point t=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.

B-Spline Curves

The Bezier-curve produced by the Bernstein basis function has limited flexibility.

- First, the number of specified polygon vertices fixes the order of the resulting polynomial which defines the curve.
- The second limiting characteristic is that the value of the blending function is nonzero for all parameter values over the entire curve.

The B-spline basis contains the Bernstein basis as the special case. The B-spline basis is non-global.

A B-spline curve is defined as a linear combination of control points Pi and B-spline basis function Ni,Ni, k (t) given by

C(t)=∑ni=0PiNi,k(t),C(t)=∑i=0nPiNi,k(t), n≥k−1,n≥k−1, tϵ[tk−1,tn+1]tϵ[tk−1,tn+1]

Where,

- {pipi: i=0, 1, 2….n} are the control points
- k is the order of the polynomial segments of the B-spline curve. Order k means that the curve is made up of piecewise polynomial segments of degree k - 1,
- the Ni,k(t)Ni,k(t) are the “normalized B-spline blending functions”. They are described by the order k and by a non-decreasing sequence of real numbers normally called the “knot sequence”.

ti:i=0,...n+Kti:i=0,...n+K

The N_i, k functions are described as follows −

Ni,1(t)={1,0,ifuϵ[ti,ti+1)OtherwiseNi,1(t)={1,ifuϵ[ti,ti+1)0,Otherwise

and if k > 1,

Ni,k(t)=t−titi+k−1Ni,k−1(t)+ti+k−tti+k−ti+1Ni+1,k−1(t)Ni,k(t)=t−titi+k−1Ni,k−1(t)+ti+k−tti+k−ti+1Ni+1,k−1(t)

and

tϵ[tk−1,tn+1)tϵ[tk−1,tn+1)

Properties of B-spline Curve

B-spline curves have the following properties −

- The sum of the B-spline basis functions for any parameter value is 1.
- Each basis function is positive or zero for all parameter values.
- Each basis function has precisely one maximum value, except for k=1.

- The maximum order of the curve is equal to the number of vertices of defining polygon.
- The degree of B-spline polynomial is independent on the number of vertices of defining polygon.
- B-spline allows the local control over the curve surface because each vertex affects the shape of a curve only over a range of parameter values where its associated basis function is nonzero.
- The curve exhibits the variation diminishing property.
- The curve generally follows the shape of defining polygon.
- Any affine transformation can be applied to the curve by applying it to the vertices of defining polygon.
- The curve line within the convex hull of its defining polygon.

2.5) FILL AREA ALGORITHMS:

In last part we have seen scan conversion of lines, circles, ellipse. In this part we are going to study polygons filling algorithms.

An inside test:

Once the polygon is entered in the display file, we can draw the outline of the polygon. To show polygon as a solid object we have to set the pixels inside the polygon as well as pixels on the boundary of it. Now the question is how to determine whether or not a point is inside of a polygon. One simple method of doing this is to construct a line segment between the point in question and a point known to be outside the polygon, as shown in the Fig. 3.6. Now count how many intersections of the line segment with the polygon boundary occur. If there are an odd number of intersections, then the point in question is inside; otherwise it is outside. This method is called the **even-odd method** of determining polygon inside points.

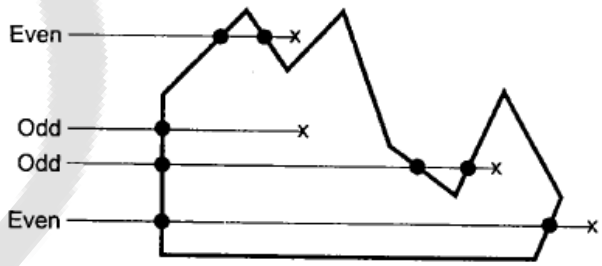
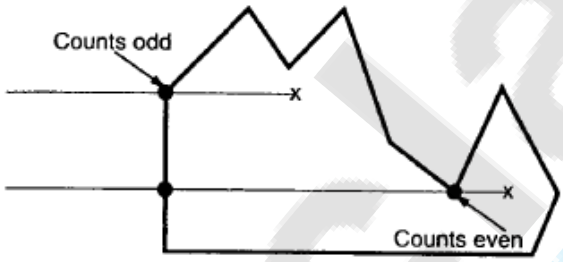


Fig. 3.6

If the intersection point is vertex of the polygon then we have to look at the other endpoints of the two segments which meet at this vertex. If these points lie on the same side of the constructed line, then the point in question counts as an even number of intersections. If they lie on opposite sides of the constructed line, then the point is counted as a single intersection. This is illustrated in Fig. 3.7.

Another approach to do the inside test is the **winding-number method**. Let us consider a point in question and a point on the polygon boundary. Conceptually, we can stretch a piece of elastic between these points and slide the end point on the polygon boundary for one complete rotation. We can then examine the point in question to see how many times the

elastic has wound around it. If it is wound at least once, then the point is inside. If there is no net winding, then the point is outside.



2.5.1) SCAN LINE POLYGON FILL ALGORITHM

3.6 Polygon Filling

Filling the polygon means highlighting all the pixels which lie inside the polygon with any colour other than background colour. Polygons are easier to fill since they have linear boundaries.

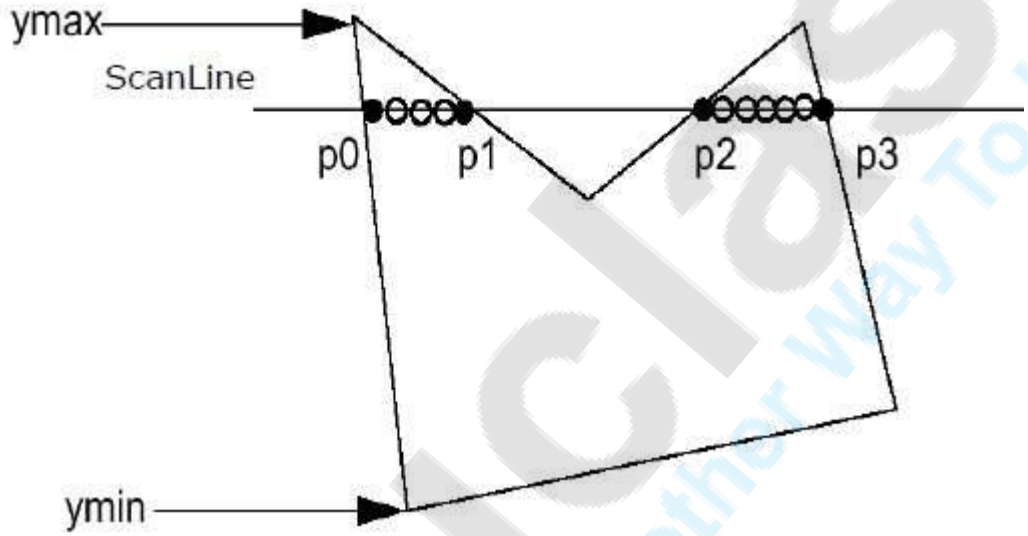
There are two basic approaches used to fill the polygon. One way to fill a polygon is to start from a given "seed", point known to be inside the polygon and highlight outward from this point i.e. neighbouring pixels until we encounter the boundary pixels. This approach is called **seed fill** because colour flows from the seed pixel until reaching the polygon boundary, like water flooding on the surface of the container. Another approach to fill the polygon is to apply the inside test i.e. to check whether the pixel is inside the polygon or outside the polygon and then highlight pixels which lie inside the polygon. This approach is known as **scan-line algorithm**. It avoids the need for a seed pixel but it requires some computation. Let us see these two methods in detail.

SCAN LINE BOUNDARY FILL ALGORITHM:

- Scan line polygon filling algorithm proceeds by scan line.

- For each scan line crossing a polygon the areas fill algorithm locates the intersection points with polygon edges.
- These intersection points are then sorted on increasing order of their x values.
- This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections.

- **Step 1** – Find out the Ymin and Ymax from the given polygon.



- **Step 2** – ScanLine intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.
- **Step 3** – Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).
- **Step 4** – Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

Scan Line Conversion Algorithm for Polygon Filling :

1. Read n, the number of vertices of polygon
2. Read x and y coordinates of all vertices in array x[n] and y[n].
3. Find y_{min} and y_{max}.
4. Store the initial x value (x_1) y values y_1 and y_2 for two endpoints and x increment Δx from scan line to scan line for each edge in the array edges [n] [4].

   While doing this check that $y_1 > y_2$, if not interchange y_1 and y_2 and corresponding x_1 and x_2 so that for each edge, y_1 represents its maximum y coordinate and y_2 represents it minimum y coordinate.
5. Sort the rows of array, edges [n] [4] in descending order of y_1, descending order of y_2 and ascending order of x_2.
6. Set $y = y_{max}$
7. Find the active edges and update active edge list :

   if ($y > y_2$ and $y \leq y_1$)

   { edge is active }

   else

   { edge is not active }
8. Compute the x intersects for all active edges for current y value [initially x-intersect is x_1 and x intersects for successive y values can be given as

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

   where $\Delta x = -\frac{1}{m}$ and $m = \frac{y_2 - y_1}{x_2 - x_1}$ i.e. slope of a line segment
9. If x intersect is vertex i.e. x-intersect = x_1 and $y = y_1$ then apply vertex test to check whether to consider one intersect or two intersects. Store all x intersects in the x-intersect [ ] array.
10. Sort x-intersect [ ] array in the ascending order,
11. Extract pairs of intersects from the sorted x-intersect [ ] array.
12. Pass pairs of x values to line drawing routine to draw corresponding line segments
13. Set $y = y - 1$
14. Repeat steps 7 through 13 until $y \geq y_{min}$.
15. Stop

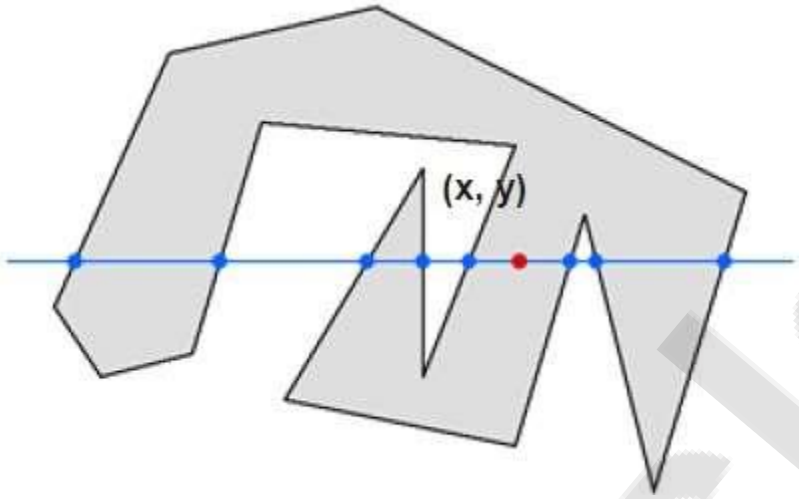
2.5.2) Inside-Outside Tests

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

- **Odd-Even Rule**
- **Nonzero winding number rule**

Odd-Even Rule

In this technique, we will count the edge crossing along the line from any point (x,y) to infinity. If the number of interactions is odd, 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. The following example depicts this concept.

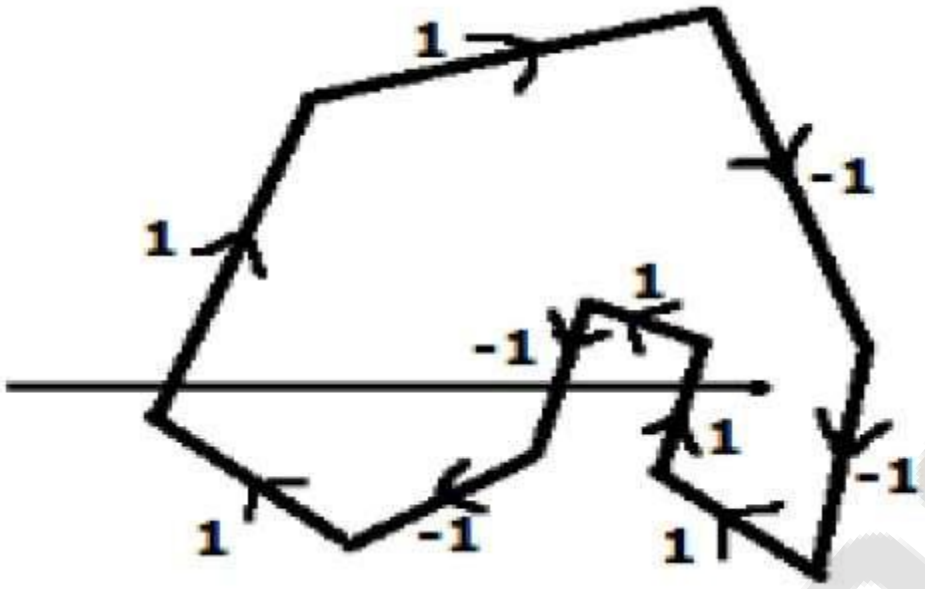


From the above figure, we can see that from the point (x,y), the number of interactions point on the left side is 5 and on the right side is 3. From both ends, the number of interaction points is odd, so the point is considered within the object.

Nonzero Winding Number Rule

This method is also used with the simple polygons to test the given point is interior or not. It can be simply understood with the help of a pin and a rubber band. Fix up the pin on one of the edge of the polygon and tie-up the rubber band in it and then stretch the rubber band along the edges of the polygon.

When all the edges of the polygon are covered by the rubber band, check out the pin which has been fixed up at the point to be test. If we find at least one wind at the point consider it within the polygon, else we can say that the point is not inside the polygon.



In another alternative method, give directions to all the edges of the polygon. Draw a scan line from the point to be test towards the left most of X direction.

- Give the value 1 to all the edges which are going to upward direction and all other -1 as direction values.
- Check the edge direction values from which the scan line is passing and sum up them.
- 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 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.

2.5.3) BOUNDARY FILL ALGORITHM:

- Boundary fill algorithm starts as a point inside a region and the point the interior portion toward the boundary.
- If the boundary is specified in a single color the algorithm proceeds outward pixel by pixel until the boundary color in encountered.
- A boundary fill algorithm accepts coordinates of an interior point, a fill color and a boundary color.

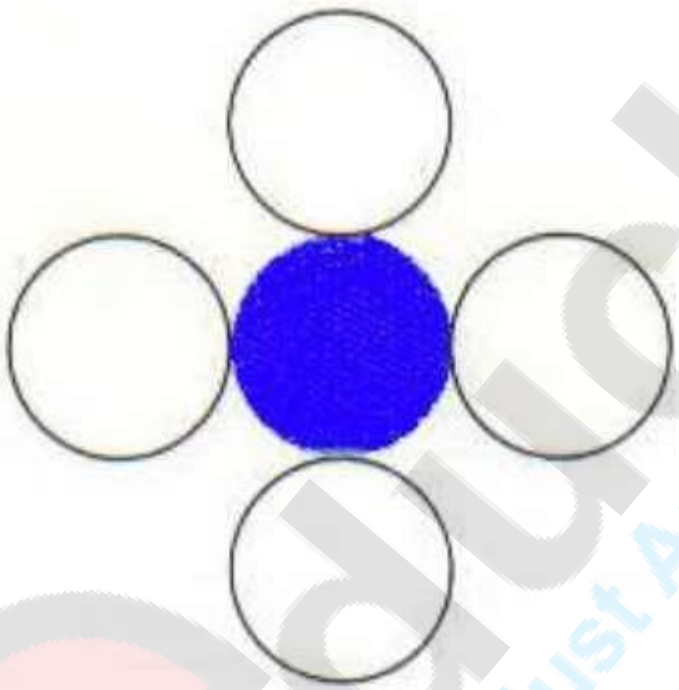
The boundary fill algorithm works as its name. This algorithm picks a point inside an object and starts to fill until it hits the boundary of the

object. The color of the boundary and the color that we fill 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 Polygon

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 will continue until we find a boundary with different color.



Algorithm

Step 1 – Initialize the value of seed point (seedx, seedy), fcolor and dcol.

Step 2 – Define the boundary values of the polygon.

Step 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
```

Step 4 – Change the default color with the fill color at the seed point.

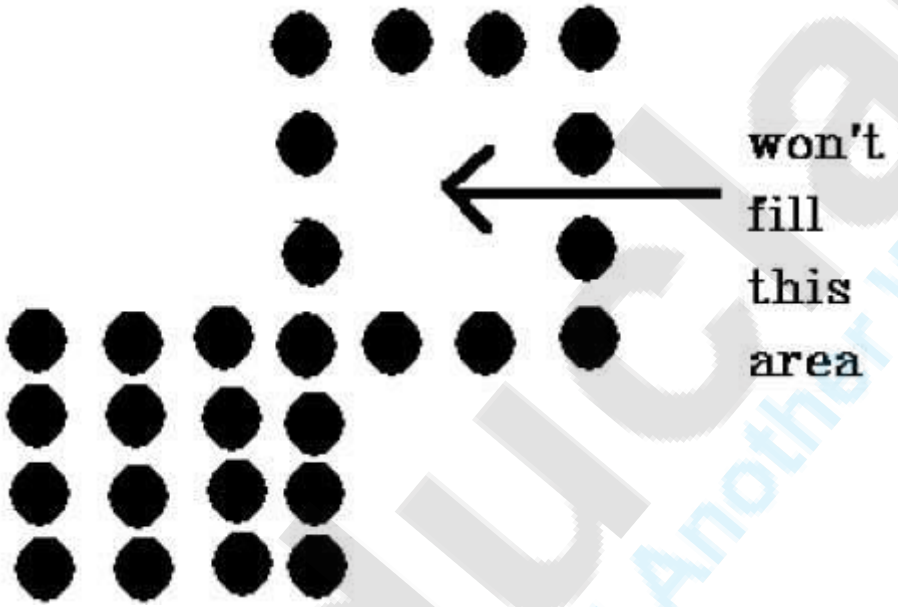
```
setPixel(seedx, seedy, fcol)
```

Step 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 – 1, seedy + 1, fcol, dcol)
```

Step 6 – Exit

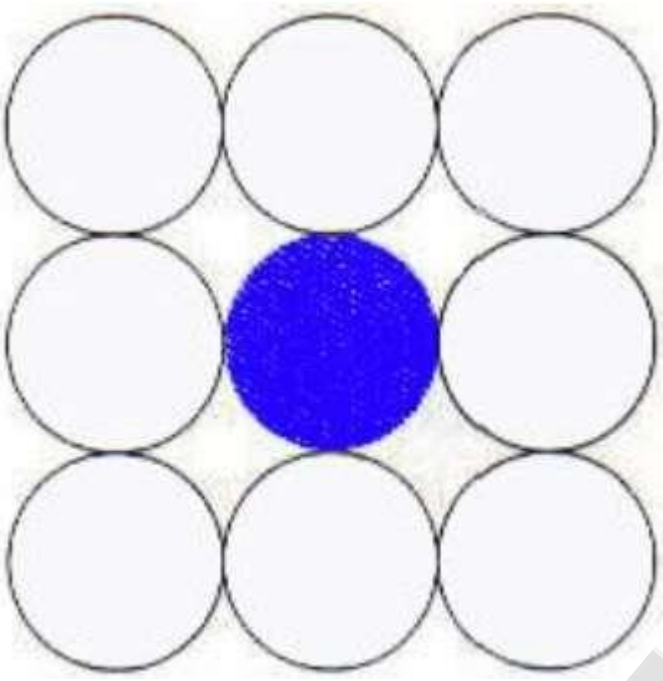
There is a problem with this technique. Consider the case as shown below where we tried to fill the entire region. Here, the image is filled only partially. In such cases, 4-connected pixels technique cannot be used.



8-Connected Polygon

In this technique 8-connected pixels are used as shown in the figure. We are putting pixels above, below, right and left side of the current pixels as we were doing in 4-connected technique.

In addition to this, we are also putting pixels in diagonals so that entire area of the current pixel is covered. This process will continue until we find a boundary with different color.



Algorithm

Step 1 – Initialize the value of seed point (seedx, seedy), fcolor and dcol.

Step 2 – Define the boundary values of the polygon.

Step 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
```

Step 4 – Change the default color with the fill color at the seed point.

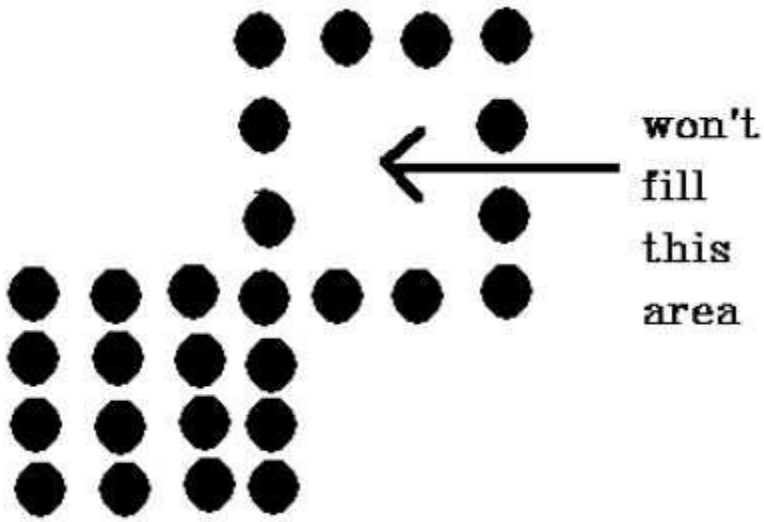
```
setPixel(seedx, seedy, fcol)
```

Step 5 – Recursively follow the procedure with four neighbourhood 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)
FloodFill (seedx – 1, seedy + 1, fcol, dcol)
FloodFill (seedx + 1, seedy + 1, fcol, dcol)
FloodFill (seedx + 1, seedy - 1, fcol, dcol)
FloodFill (seedx – 1, seedy - 1, fcol, dcol)
```

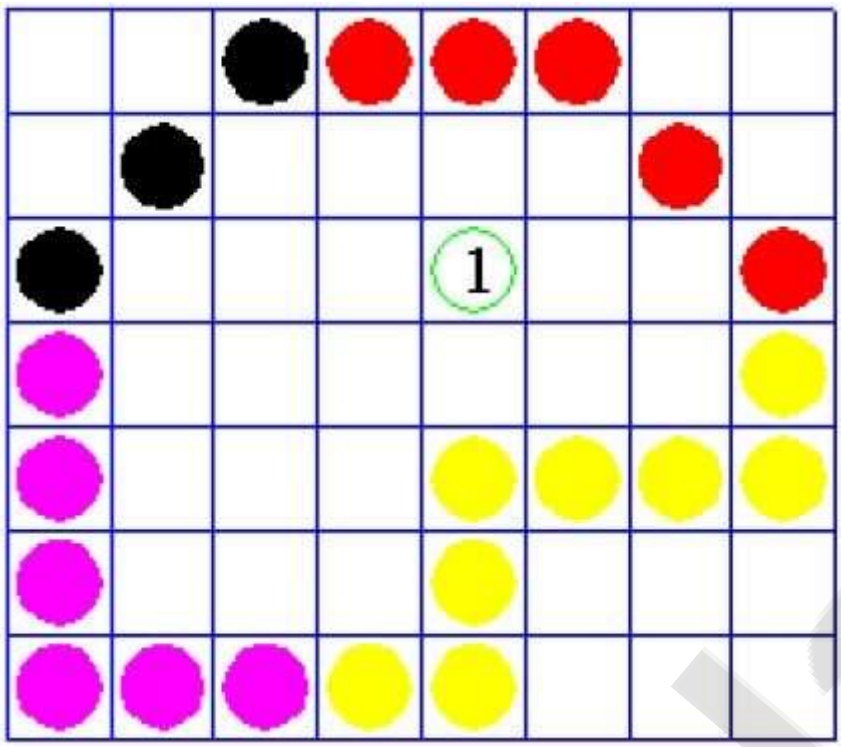
Step 6 – Exit

The 4-connected pixel technique failed to fill the area as marked in the following figure which won't happen with the 8-connected technique.



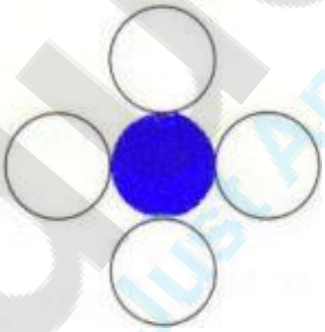
2.5.4) FLOOD FILL ALGORITHM

- 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.
- Once again, this algorithm relies on the Four-connect or Eight-connect method of filling in the pixels. But instead of looking for the boundary color, it is looking for all adjacent pixels that are a part of the interior.



For this purpose we can create a function or we can use a predefined function in the graphics.h header file which takes 3 arguments:-

floodfill(x,y,color)



4 Connected Region (Image Source)

In Flood Fill algorithm we start with some seed and examine the neighboring pixels, however pixels are checked for a specified interior color instead of boundary color and is replaced by a new color. It can be done using 4 connected or 8 connected region method.

Below we use 4 connected region recursive algorithm to implement this algorithm.

Algorithm

1. Create a function called as **floodFill (x,y,oldcolor,newcolor)**

```
void floodFill(int x,int y,int oldcolor,int newcolor)
{
    if(getpixel(x,y) == oldcolor)
    {
        putpixel(x,y,newcolor);
        floodFill(x+1,y,oldcolor,newcolor);
        floodFill(x,y+1,oldcolor,newcolor);
        floodFill(x-1,y,oldcolor,newcolor);
        floodFill(x,y-1,oldcolor,newcolor);
    }
}
//getpixel(x,y) gives the color of specified pixel
```

2. Repeat until the polygon is completely filled.

3. Stop.

C++ Program for flood fill algorithm

```
#include<iostream.h>
#include<graphics.h>
#include<dos.h>

void floodFill(int x,int y,int oldcolor,int
newcolor)
{
    if(getpixel(x,y) == oldcolor)
    {
        putpixel(x,y,newcolor);
        floodFill(x+1,y,oldcolor,newcolor);
        floodFill(x,y+1,oldcolor,newcolor);
        floodFill(x-1,y,oldcolor,newcolor);
        floodFill(x,y-1,oldcolor,newcolor);
    }
}
//getpixel(x,y) gives the color of
specified pixel

int main()
{
    int gm,gd=DETECT,radius;
    int x,y;
```

```
    cout<<"Enter x and y positions for
circle\n";
    cin>>x>>y;
    cout<<"Enter radius of circle\n";
    cin>>radius;

    initgraph(&gd,&gm,"c:\\turboc3\\bgi");
    circle(x,y,radius);
    floodFill(x,y,0,15);
    delay(5000);
    closegraph();

    return 0;
}
```

Eduklash Just Another Way To Learn

UNIT 3

2D GEOMETRIC TRANSFORMATION AND CLIPPING

3.1) BASIC TRANSFORMATION:

2d transformation:

There are 3 types of 2d transformations:

1. Translation
2. Scaling
3. Rotation

Translation:

Translation is a process of changing the position of an object in a straight line path from one coordinate location to another.

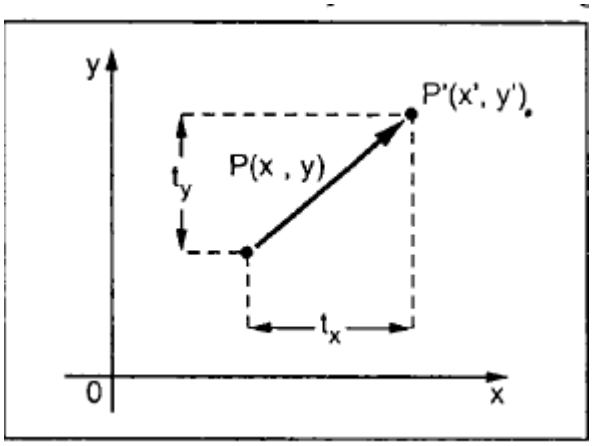
We can translate a 2 dimensional point by adding translation distance **tx and ty** to the original coordinate position(x, y) to move the point to a new position (x', y').

X'=x+ tx

Y'=y+ ty

The translation distance pair (tx, ty) is called as **translation vector or shift vector.**

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



The two dimensional translation equation In the matrix form is:

P′=P+T

Ex. 4.1 : *Translate a polygon with coordinates A (2, 5), B (7, 10) and C (10, 2) by 3 units in x direction and 4 units in y direction.*

Sol. :

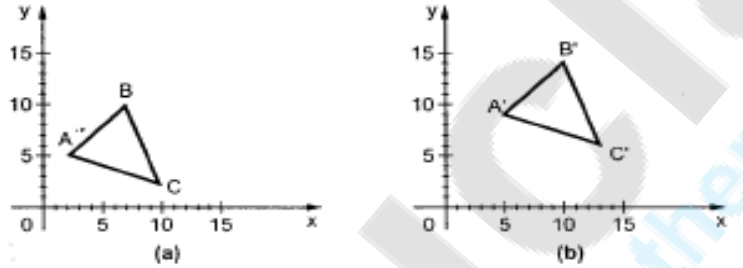


Fig. 4.2 Translation of polygon

$$A' = A + T$$
$$= \begin{bmatrix} 2 \\ 5 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$
$$= \begin{bmatrix} 5 \\ 9 \end{bmatrix}$$

$$B' = B + T$$
$$= \begin{bmatrix} 7 \\ 10 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$
$$= \begin{bmatrix} 10 \\ 14 \end{bmatrix}$$

$$C' = C + T$$
$$= \begin{bmatrix} 10 \\ 2 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$
$$= \begin{bmatrix} 13 \\ 6 \end{bmatrix}$$

ROTATION:

A 2d rotation is applied to an object by repositioning it along a circular path in the xy plane.

To generate a rotation we specify a rotation angle and the position about which the object is to rotated.

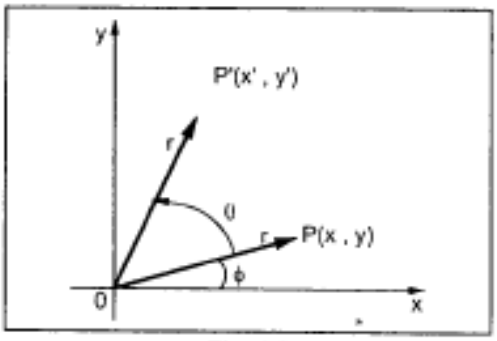


Fig. 4.3

Let us consider the rotation of the object about the origin, as shown in the Fig. 4.3.

Here, r is the constant distance of the point from the origin, angle ϕ is the original angular position of the point from the horizontal, and θ is the rotation angle. Using standard trigonometric equations, we can express the transformed coordinates in terms of angles θ and ϕ as

$$\left.\begin{aligned} x' &= r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ y' &= r\sin(\phi+\theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta \end{aligned}\right\} \quad \ldots(4.4)$$

The original coordinates of the point in polar coordinates are given as

$$\left.\begin{aligned} x &= r\cos\phi \\ y &= r\sin\phi \end{aligned}\right\} \quad \ldots(4.5)$$

Substituting equations 4.5 into 4.4, we get the transformation equations for rotating a point (x, y) through an angle θ about the origin :

$$\left.\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \end{aligned}\right\} \quad \ldots(4.6)$$

The above equations can be represented in the matrix form as given below

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

$$\therefore \quad P' = P \cdot R \quad \ldots(4.7)$$

where R is rotation matrix and it is given as

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \quad \ldots(4.8)$$

It is important to note that positive values for the rotation angle define counterclockwise rotations about the rotation point and negative values rotate objects in the clockwise sense.

For negative values of θ i.e., for clockwise rotation, the rotation matrix becomes

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

$$= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad \because \cos(-\theta) = \cos\theta \text{ and } \sin(-\theta) = -\sin\theta \quad \ldots(4.9)$$

Ex. 4.2 : *A point (4, 3) is rotated counterclockwise by an angle of 45°. Find the rotation matrix and the resultant point.*

Sol. :

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} \cos 45^\circ & \sin 45^\circ \\ -\sin 45^\circ & \cos 45^\circ \end{bmatrix}$$

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

$$\therefore \quad P^1 = [4\ 3] \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

$$= \left[4/\sqrt{2} - 3/\sqrt{2} \quad 4/\sqrt{2} + 3/\sqrt{2} \right]$$

$$= \left[1/\sqrt{2} \quad 7/\sqrt{2} \right]$$

SCALING:

A scaling transformation changes the size of an object. This operation can be carried out for polygons by multiplying the coordinate value(x, y) of each vertex by scaling factor Sx and Sy to produce the transformed coordinates(x', y').

X'=x.Sx

Y'=y.Sy

Scaling factor S_x scales object in the x direction and scaling factor S_y scales object in the y direction. The equations 4.10 can be written in the matrix form as given below :

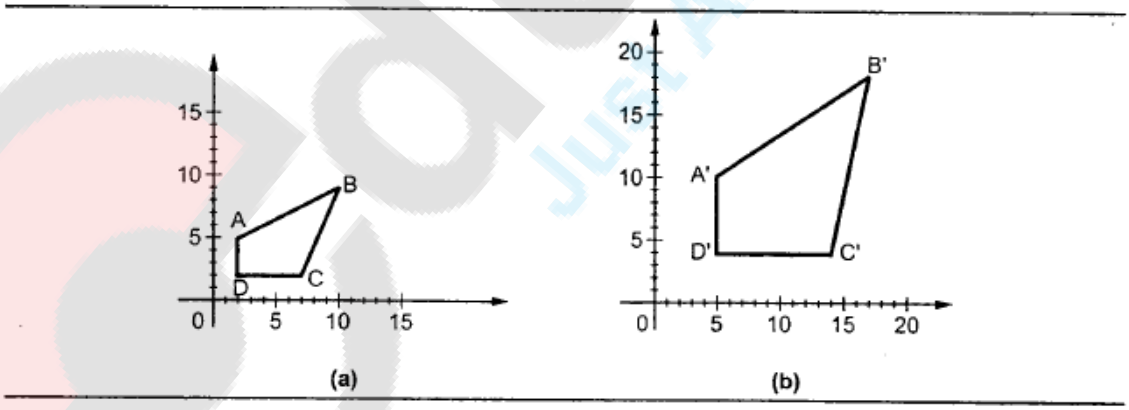


Fig. 4.4

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

$$= [x \cdot S_x \quad y \cdot Sy] \qquad \ldots (4.11)$$

$$= P \cdot S$$

Any positive numeric values are valid for scaling factors Sx, Sy. Values less than 1 reduces the size of the object and values greater than 1 produce an enlarged object. For both Sx and Sy value equal to 1 the size of the object doesnot change.

Ex. 4.3 : *Scale the polygon with coordinates A (2, 5), B (7, 10) and C (10, 2) by two units in x direction and two units in y direction.*

Sol. : Here $S_x = 2$ and $S_y = 2$. Therefore, transformation matrix is given as

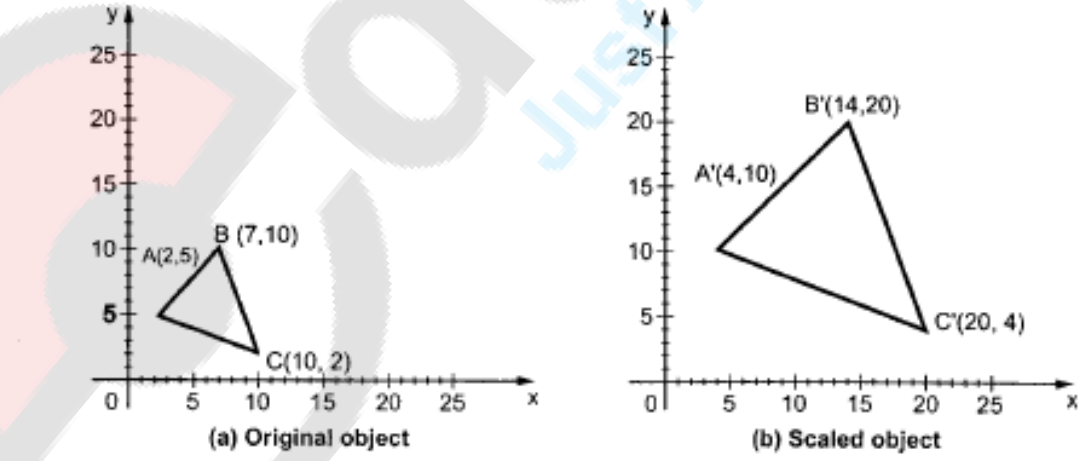
$$S = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

The object matrix is :

$$\begin{matrix} & x & y \\ A & 2 & 5 \\ B & 7 & 10 \\ C & 10 & 2 \end{matrix}$$

$$\therefore \quad \begin{matrix} A' \\ B' \\ C' \end{matrix} \begin{bmatrix} x'_1 & y'_1 \\ x'_2 & y'_2 \\ x'_3 & y'_3 \end{bmatrix} = \begin{bmatrix} 2 & 5 \\ 7 & 10 \\ 10 & 2 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 10 \\ 14 & 20 \\ 20 & 4 \end{bmatrix}$$



3.2) HOMOGENOUS COORDINATES:

In design and picture formation process, many times we may require to perform translation, rotations, and scaling to fit the picture components into their proper positions. In the previous section we have seen that each of the basic transformations can be expressed in the general matrix form

$$P' = P \cdot M_1 + M_2 \quad \ldots (4.12)$$

For translation :

$$P' = P \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

i.e. M_1 = Identity matrix

M_2 = Translation vector

For rotation :

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

i.e. M_1 = Rotational matrix

$M_2 = 0$

For scaling :

$$P' = P \cdot \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

i.e. M_1 = Scaling matrix

$M_2 = 0$

Eduklash Just Another Way To Learn

4.3.1 Homogeneous Coordinates for Translation

The homogeneous coordinates for translation are given as

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

Therefore, we have

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

$$= [x + t_x \quad y + t_y \quad 1]$$

4.3.2 Homogeneous Coordinates for Rotation

The homogeneous coordinates for rotation are given as

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

Therefore, we have

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

$$= [x\cos\theta - y\sin\theta \quad x\sin\theta + y\cos\theta \quad 1]$$

4.3.3 Homogeneous Coordinates for Scaling

The homogeneous coordinate for scaling are given as

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

Therefore, we have

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

$$= [x \cdot S_x \quad y \cdot S_y \quad 1]$$

Ex. 4.4 : *Give a 3 × 3 homogeneous coordinate transformation matrix for each of the following translations*

a) Shift the image to the right 3-units

b) Shift the image up 2 units

c) Move the image down ½ unit and right 1 unit

d) Move the image down 2/3 unit and left 4 units

Sol. : We know that homogenous coordinates for translation are

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

a) Here, $t_x = 3$ and $t_Y = 0$

$$\therefore \quad T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix}$$

b) Here, $t_x = 0$ and $t_y = 2$

$$\therefore \quad T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix}$$

c) Here, $t_x = 1$ and $t_y = -0.5$

$$\therefore \quad T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -0.5 & 1 \end{bmatrix}$$

d) Here, $t_x = -4$ and $t_y = -0.66$

$$\therefore \quad T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -0.66 & 1 \end{bmatrix}$$

Eduklash Just Another Way To Learn

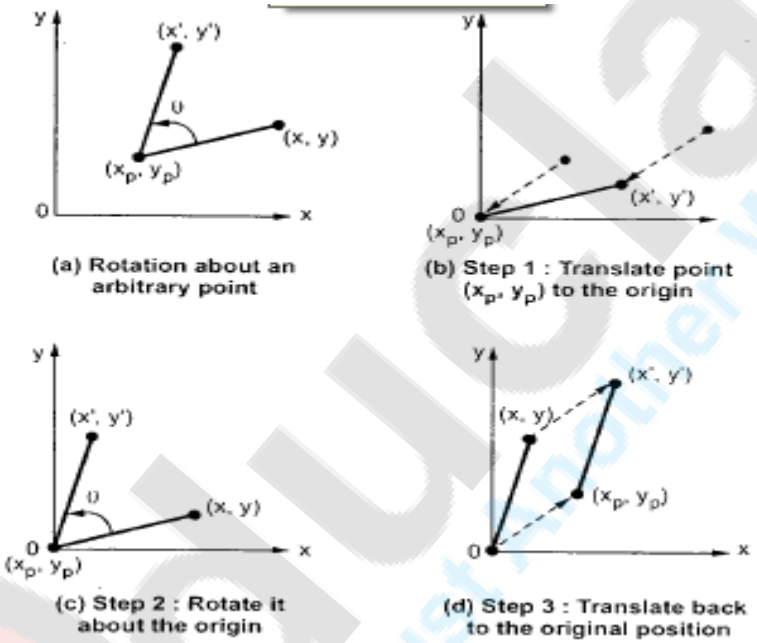
3.3) COMPOSITION of 2D TRANSFORMATION:

The basic purpose of composite transformation is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformation one after the other.

Rotation about Arbitrary Point:

To rotate an object about an arbitrary point (xp,yp) we have to carry out three step:

1. Translate point (xp, yp) to the origin.
2. Rotate it about the origin
3. Finally translate the centre of the rotation back where it belong.



The translation matrix to move point (x_p, y_p) to the origin is given as

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_p & -y_p & 1 \end{bmatrix}$$

The rotation matrix for counterclockwise rotation of point about the origin is given as

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

The translation matrix to move the center point back to its original position is given as

$$T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_p & y_p & 1 \end{bmatrix}$$

Therefore, the overall transformation matrix for a counterclockwise rotation by an angle θ about the point (x_p, y_p) is given as

$$T_1 \cdot R \cdot T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_p & -y_p & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_p & y_p & 1 \end{bmatrix}$$

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

$$= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_p\cos\theta + y_p\sin\theta + x_p & -x_p\sin\theta - y_p\cos\theta + y_p & 1 \end{bmatrix} \quad \ldots (4.18)$$

TRANSFORMATION:

The three basic transformations of scaling, rotating, and translating are the most useful and most common. There are some other transformations which are useful in certain applications. Two such transformations are reflection and shear.

4.5.1 Reflection

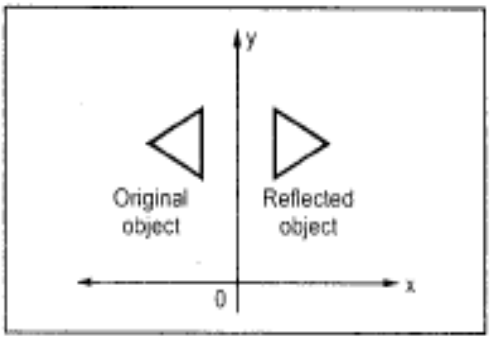
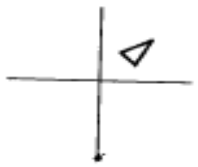
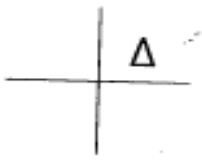
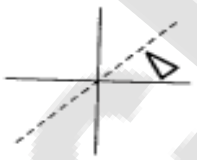
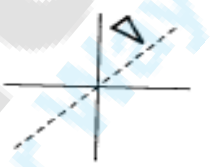
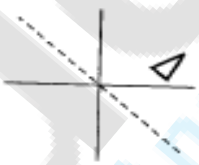


Fig. 4.7 Reflection about y axis

A reflection is a transformation that produces a mirror image of an object relative to an axis of reflection. We can choose an axis of reflection in the xy plane or perpendicular to the xy plane. The table 4.1 gives examples of some common reflections.

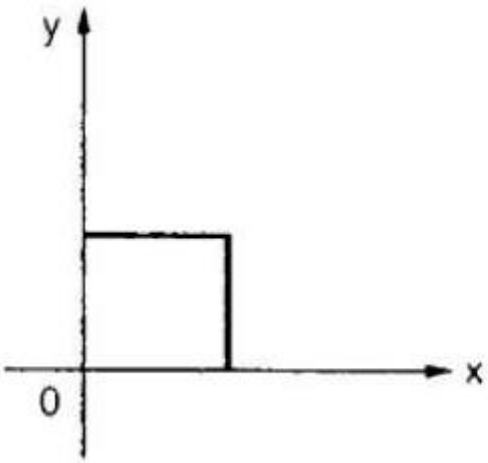
Reflection	Transformation matrix	Original image	Reflected image
Reflection about Y-axis	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Reflection about X axis	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Reflection about origin	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Reflection about line $y = x$	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Reflection about line $y = -x$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		

3.4) Shear

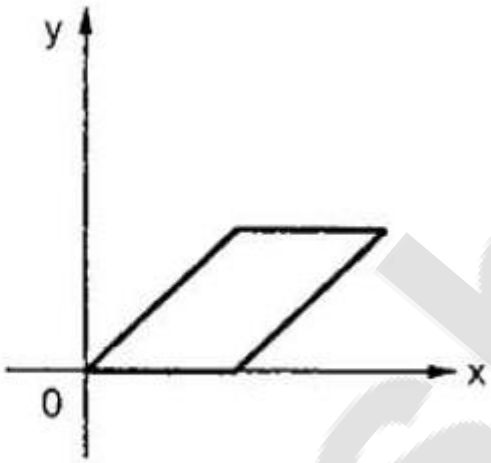
A transformation that slants the shape of an object is called the shear transformation. There are two shear transformations **X-Shear** and **Y-Shear**. One shifts X coordinates values and other shifts Y coordinate values. However; in both the cases only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as **Skewing**.

X-Shear

The X-Shear preserves the Y coordinate and changes are made to X coordinates, which causes the vertical lines to tilt right or left as shown in below figure.



(a) Original object



(b) Object after x shear

The transformation matrix for X-Shear can be represented as –

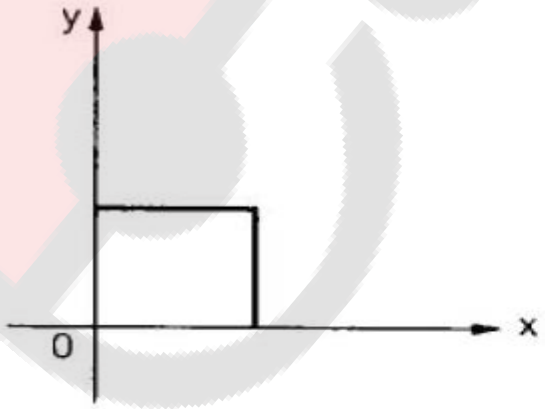
$$X_{sh} = \begin{bmatrix} 1 & 0 & 0 \\ shx & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$X' = X + Sh_X \cdot Y$$

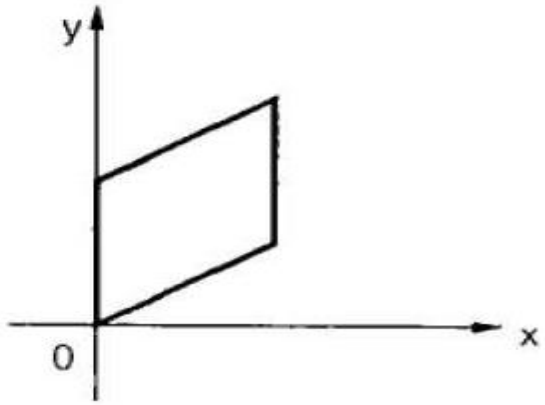
$$Y' = Y$$

Y-Shear

The Y-Shear preserves the X coordinates and changes the Y coordinates which causes the horizontal lines to transform into lines which slopes up or down as shown in the following figure.



(a) Original object



(b) Object after y shear

The Y-Shear can be represented in matrix from as –

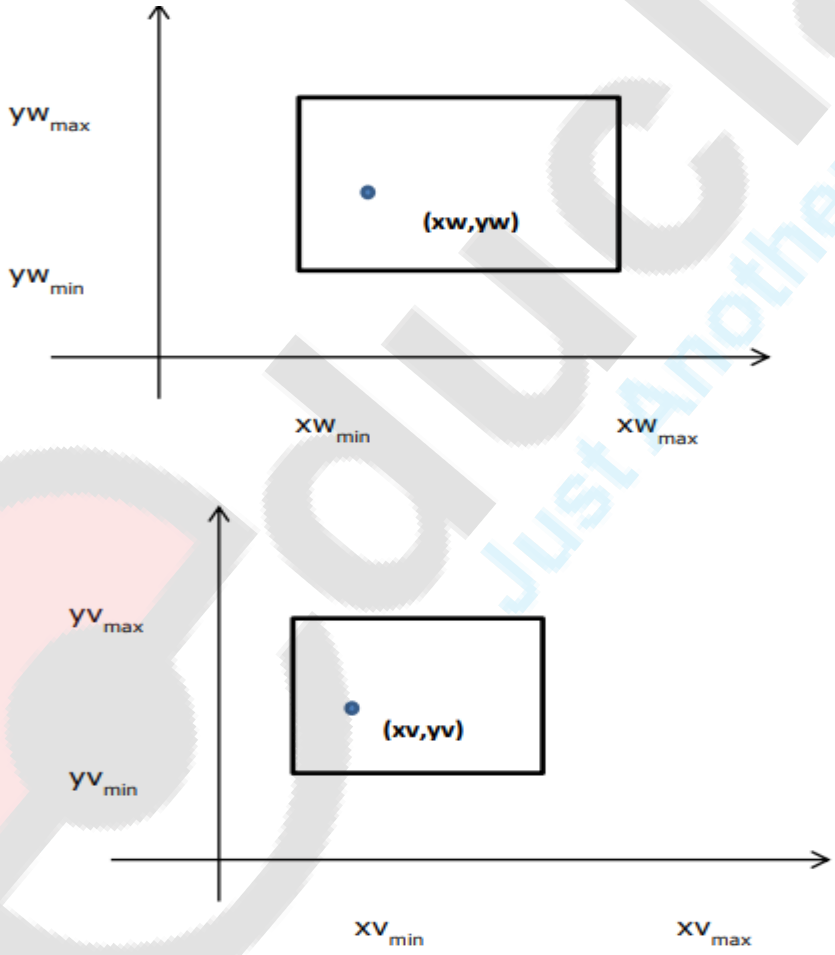
$$Y_{sh} \begin{bmatrix} 1 & shy & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Y' = Y + Sh_y \cdot X$$

$$X' = X$$

3.5) WINDOW TO VIEWPORT:

A world-coordinate area selected for display is called a window. An area on a display device to which a window is mapped is called a viewport. The window defines what is to be viewed; the viewport defines where it is to be displayed.



(Figure . A point at position (xw,yw) in a designated window is mapped to viewport coordinates (xv, yv). So that relative positions in the two areas are the same.)

Figure illustrates the window-to-viewport mapping. A point at position (xw,yw) in the window is mapped into position (xv, yv) in the associated viewport. To maintain the same relative placement in the viewport as in the window, we require that

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

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

Solving these expressions for the viewport position (xv, yv), we have

$$xv = xv_{min} + (xw - xw_{min})sx$$

$$yv = yv_{min} + (yw - yw_{min})sy$$

where the scaling factors are

$$sx = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

$$sx = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

Equations A can also be derived with a set of transformations that converts the window area into the viewport area.

This conversion is performed with the following sequence of transformations:

1. Perform a scaling transformation using a fixed-point position of (xwmin, ywmin) that scales the window area to the size of the viewport.

2. Translate the scaled window area to the position of the viewport. Relative proportions of objects are maintained if the scaling factors are the same (sx = sy). Otherwise, world objects will be stretched or contracted in either the x or y direction when displayed on the output device.

3.7) CLIPPING

Many graphics application programs give the users the impression of looking through a window at a very large picture. Figure shows the use of this effect in a program for viewing different portions of a large architectural plan at different scales. Viewing an architectural plan through windows of different sizes

This makes use of scaling and translation techniques to generate a variety of different views of a single representation of a plan.

To display an enlarged portion of a picture, we must not only apply the appropriate scaling and translation but also should identify the visible parts of the picture. This is not straightforward. Certain lines may lie partly inside the visible

portion of the picture and partly outside. We cannot display each of these lines in its entirety.

3.7.1)POINT CLIPPING:

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

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

$$yw_{min} \leq y \leq yw_{max}$$

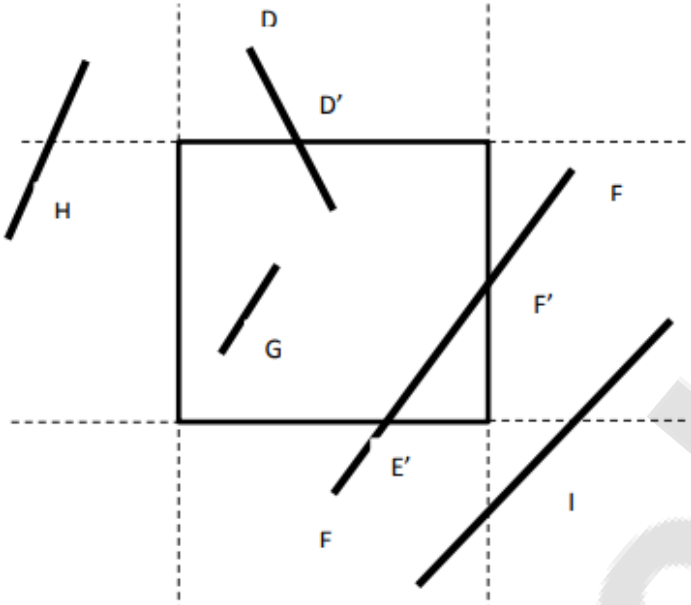
Where the edges of the clip window (xw_{min}, xw_{max}, yw_{min} , yw_{max}) can be either the world-coordinate window boundaries or viewport boundaries. If any one of these four inequalities is not satisfied, the point is clipped (not saved for display).

Although point clipping is applied less often than line or polygon clipping, some applications may require a point clipping procedure. For example, point clipping can be applied to scenes involving explosions or sea foam that are modeled with particles (points) distributed in some region of the scene.

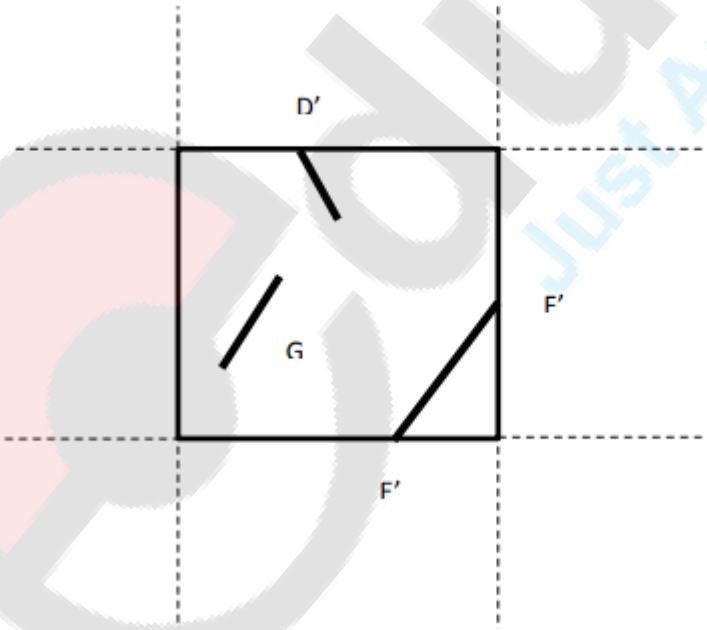
3.7.2)LINE CLIPPING:

Lines intersecting a rectangular clip region are always clipped to a single line segment. Figure shows examples of clipped lines.

Before clipping



After clipping



3.5.1) COHEN – SUTHRLAND LINE CLIPPING

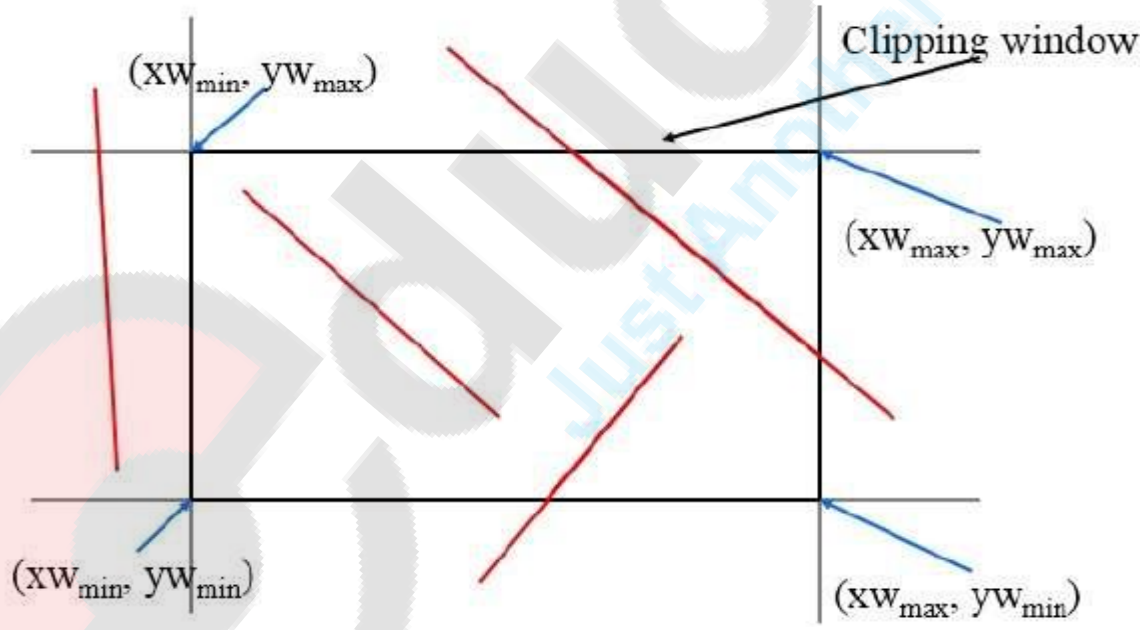
This is one of the oldest and most popular line clipping algorithm. To speed up the process this algorithm performs initial tests that reduce number of intersections that must be calculated. It does so by using a 4-bit code called as region code or outcodes. These codes identify location of the end point of line.

Each bit position indicates a direction, starting from the rightmost position of each bit indicates left, right, bottom, top respectively.

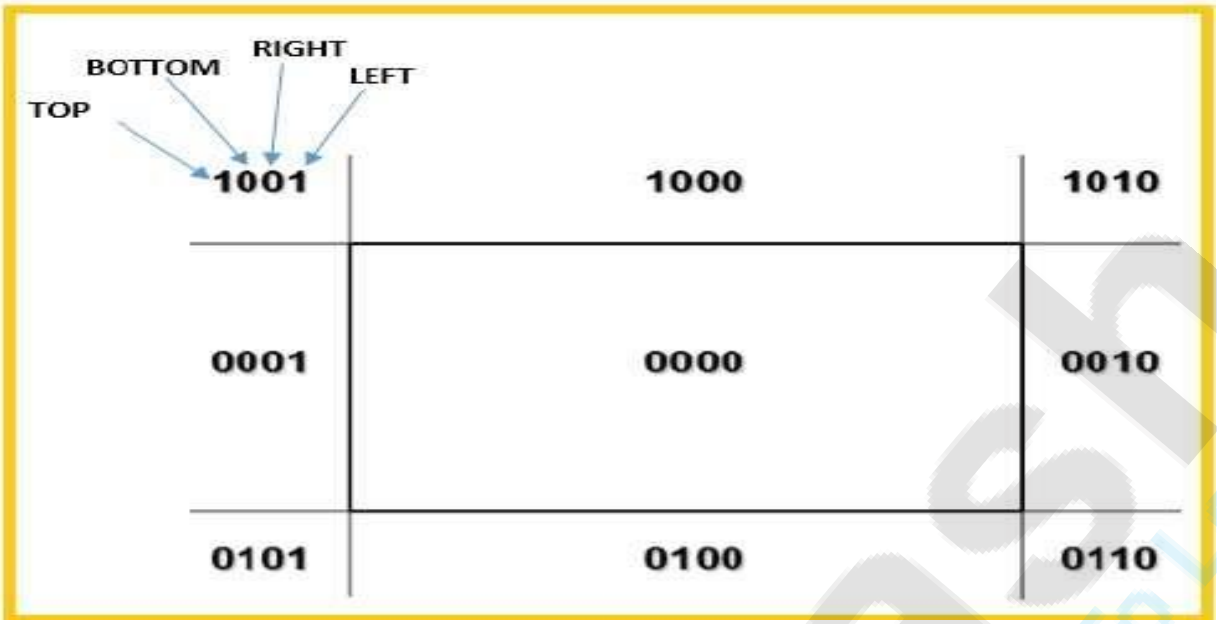
Cohen-Sutherland algorithm divides a two-dimensional space into 9 regions and then efficiently determines the lines and portions of lines that are inside the given rectangular area.

Once we establish region codes for both the endpoints of a line we determine whether the endpoint is visible, partially visible or invisible with the help of ANDing of the region codes.

This algorithm uses the clipping window as shown in the following figure. The minimum coordinate for the clipping region is (XWmin,YWmin)(XWmin,YWmin) and the maximum coordinate for the clipping region is (XWmax,YWmax)(XWmax,YWmax).



We will use 4-bits to divide the entire region. These 4 bits represent the Top, Bottom, Right, and Left of the region as shown in the following figure. Here, the **TOP** and **LEFT** bit is set to 1 because it is the **TOP-LEFT** corner.

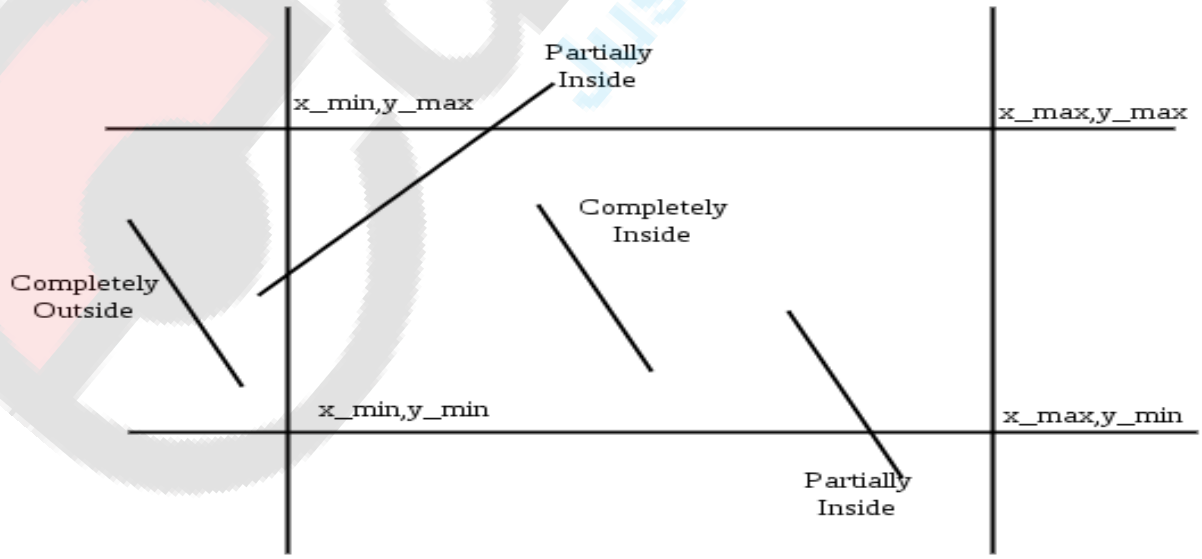


There are 3 possibilities for the line −

1. **Completely inside the given rectangle:** Bitwise OR of region of two end points of line is 0 (Both points are inside the rectangle)

2. **Completely outside the given rectangle:** Both endpoints share at least one outside region which implies that the line does not cross the visible region. (bitwise AND of endpoints != 0).

3. **Partially inside the window:** Both endpoints are in different regions. In this case, the algorithm finds one of the two points that is outside the rectangular region. The intersection of the line from outside point and rectangular window becomes new corner point and the algorithm repeats



ALGORITHM: -

- **Step 1** – Assign a region code for each endpoints.
- **Step 2** – If both endpoints have a region code **0000** then accept this line.
- **Step 3** – Else, perform the logical **AND** operation for both region codes.
- **Step 3.1** – If the result is not **0000,** then reject the line.
- **Step 3.2** – Else you need clipping.
  - **Step 3.2.1** – Choose an endpoint of the line that is outside the window.
  - **Step 3.2.2** – Find the intersection point at the window boundary (base on region code).
  - **Step 3.2.3** – Replace endpoint with the intersection point and update the region code.
  - **Step 3.2.4** – Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.
- **Step 4** – Repeat step 1 for other lines.
-

C++ Code for Cohen-Sutherland Line clipping

```
#include<iostream.h>
#include<stdlib.h>
#include<math.h>
#include<graphics.h>
#include<dos.h>

typedef struct coordinate
{
int x,y;
char code[4];
}PT;

void drawwindow();
```

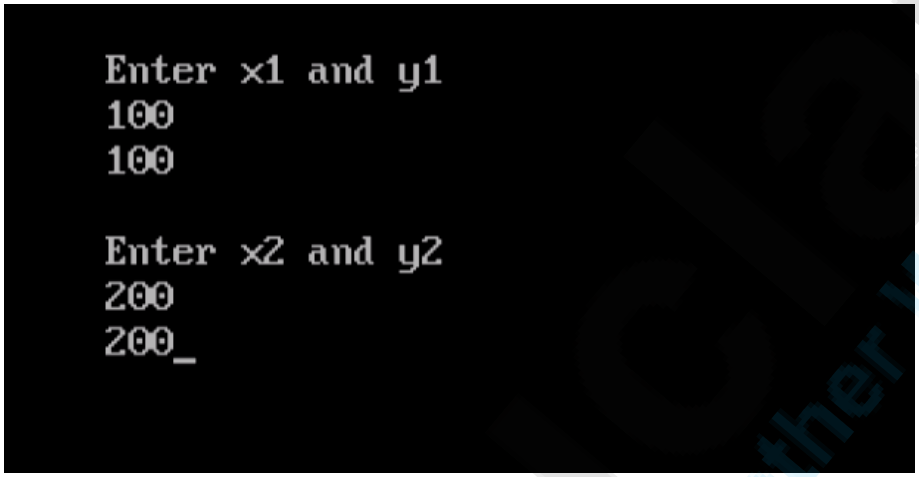
```
void drawline(PT p1,PT p2);
PT setcode(PT p);
int visibility(PT p1,PT p2);
PT resetendpt(PT p1,PT p2);

void main()
{
int gd=DETECT,v,gm;
PT p1,p2,p3,p4,ptemp;

cout<<"\nEnter x1 and y1\n";
cin>>p1.x>>p1.y;
cout<<"\nEnter x2 and y2\n";
cin>>p2.x>>p2.y;

initgraph(&gd,&gm,"c:\\turboc3\\bgi");
drawwindow();
delay(500);

drawline(p1,p2);
delay(500);
cleardevice();

delay(500);
p1=setcode(p1);
p2=setcode(p2);
v=visibility(p1,p2);
delay(500);

switch(v)
{
case 0: drawwindow();
delay(500);
drawline(p1,p2);
break;
case 1:   drawwindow();
delay(500);
break;
case 2:   p3=resetendpt(p1,p2);
p4=resetendpt(p2,p1);
drawwindow();
delay(500);
drawline(p3,p4);
break;
}

delay(5000);
closegraph();
}

void drawwindow()
```

```
{
line(150,100,450,100);
line(450,100,450,350);
line(450,350,150,350);
line(150,350,150,100);
}

void drawline(PT p1,PT p2)
{
line(p1.x,p1.y,p2.x,p2.y);
}

PT setcode(PT p)    //for setting the 4 bit code
{
PT ptemp;

if(p.y<100)
ptemp.code[0]='1';    //Top
else
ptemp.code[0]='0';

if(p.y>350)
ptemp.code[1]='1';    //Bottom
else
ptemp.code[1]='0';

if(p.x>450)
ptemp.code[2]='1';    //Right
else
ptemp.code[2]='0';

if(p.x<150)
ptemp.code[3]='1';    //Left
else
ptemp.code[3]='0';

ptemp.x=p.x;
ptemp.y=p.y;

return(ptemp);
}

int visibility(PT p1,PT p2)
{
int i,flag=0;

for(i=0;i<4;i++)
{
if((p1.code[i]!='0') || (p2.code[i]!='0'))
flag=1;
}
```

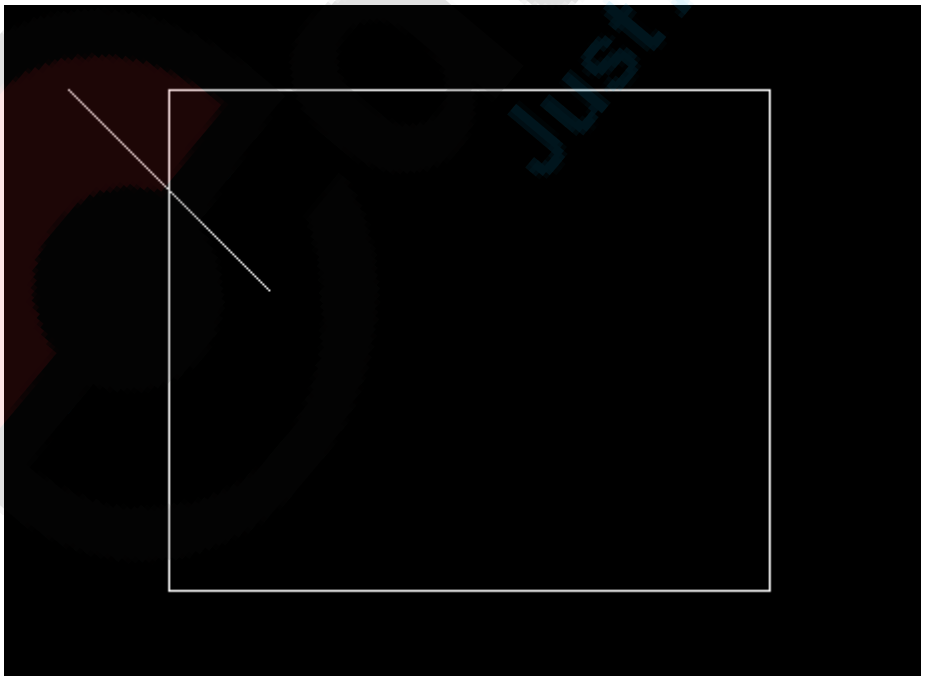
```
if(flag==0)
return(0);

for(i=0;i<4;i++)
{
if((p1.code[i]==p2.code[i]) && (p1.code[i]=='1'))
flag='0';
}

if(flag==0)
return(1);

return(2);
}

PT resetendpt(PT p1,PT p2)
{
PT temp;
int x,y,i;
float m,k;

if(p1.code[3]=='1')
x=150;

if(p1.code[2]=='1')
x=450;

if((p1.code[3]=='1') || (p1.code[2]=='1'))
{
m=(float)(p2.y-p1.y)/(p2.x-p1.x);
k=(p1.y+(m*(x-p1.x)));
temp.y=k;
temp.x=x;

for(i=0;i<4;i++)
temp.code[i]=p1.code[i];

if(temp.y<=350 && temp.y>=100)
return (temp);
}

if(p1.code[0]=='1')
y=100;

if(p1.code[1]=='1')
y=350;

if((p1.code[0]=='1') || (p1.code[1]=='1'))
{
m=(float)(p2.y-p1.y)/(p2.x-p1.x);
```

```
k=(float)p1.x+(float)(y-p1.y)/m;
temp.x=k;
temp.y=y;

for(i=0;i<4;i++)
temp.code[i]=p1.code[i];

return(temp);
}
else
return(p1);
}
```

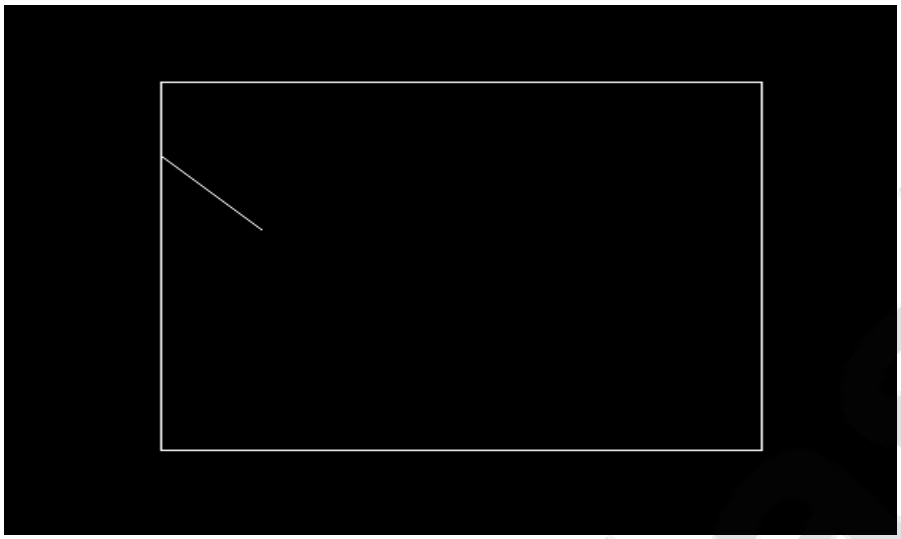
Output: -



Before Clipping: -



After Clipping: -



3.5.2) MIDPOINT SUBDIVISON ALGORITHM

Midpoint subdivision algorithm is an extension of the Cyrus Beck algorithm. This algorithm is mainly used to compute visible areas of lines that are present in the view port are of the sector or the image. It follows the principle of the bisection method and works similarly to the Cyrus Beck algorithm by bisecting the line in to equal halves. But unlike the Cyrus Beck algorithm, which only bisects the line once, Midpoint Subdivision Algorithm bisects the line numerous times.

Also, the Sutherland Cohen subdivision line clipping algorithm requires the calculation of the intersection of the line with the window edge. These calculations can be avoided by repetitively subdividing the line at its midpoint.

Like other algorithm, initially the line is tested for visibility. If line is completely visible it is drawn and if it is completely invisible it is rejected. If line is partially visible then it is subdivided in two equal parts. The visibility tests are then applied to each half. This subdivision process is repeated until we get completely visible and completely invisible line segments. This is illustrated in figure below

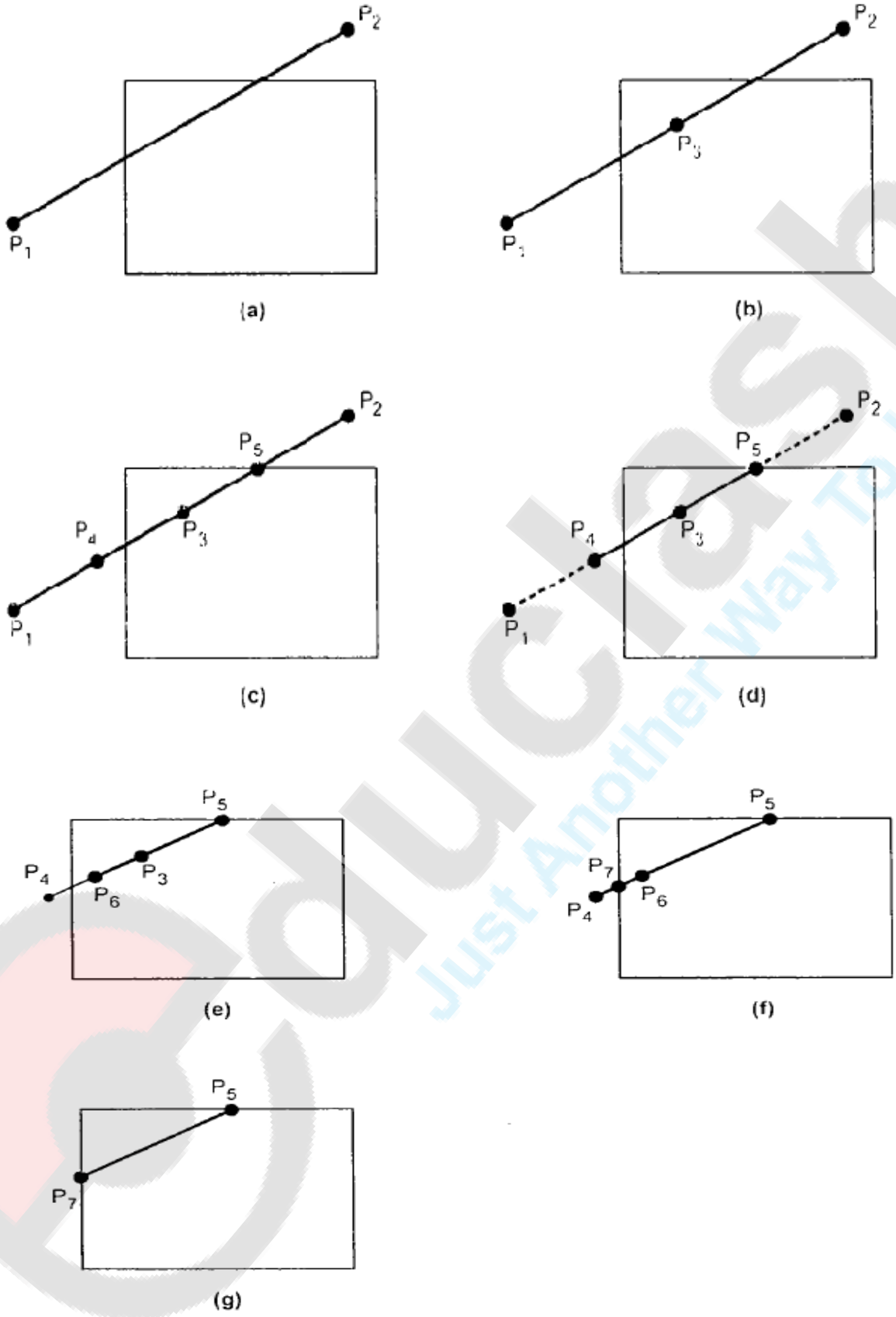


Fig. (k) Clipping line with midpoint subdivision algorithm

As shown in the figure (k), line P1 P2 is partially visible. It is subdivided in two equal Parts P1 P3 and P3 P2 (see Fig. k (b)). Both the line segments are tested for visibility and found to be partially visible. Both line segments are then subdivided in two equal parts to get midpoints P4 and P5 (see Fig. k (c)). It is observed that line segments P1 P4 and P5 P2 are completely invisible and hence rejected. However, line segment P3 P5 is completely visible and hence drawn. The remaining line segment P4 P3 is still partially visible. It is then subdivided to get midpoint P6. It is observed that P6 P3 is completely visible whereas P4 P6 is partially visible. Thus, P6 P3 line segment is drawn and P4 P6 line segment is further subdivided into equal parts to get midpoint P7. Now, it is observed that line segment P4 P7 is completely invisible and line segment P7 P6 is completely visible (see Fig. k (f)), and there is no further partially visible segment.

Midpoint Subdivision Algorithm :

1. Read two endpoints of the line say $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$.
2. Read two corners (left-top and right-bottom) of the window, say (Wx_1, Wy_1 and Wx_2, Wy_2).
3. Assign region codes for two end points using following steps :
   Initialize code with bits 0000
   Set Bit 1 - if ($x < Wx_1$)
   Set Bit 2 - if ($x > Wx_2$)
   Set Bit 3 - if ($y < Wy_1$)
   Set Bit 4 - if ($y > Wy_2$)
4. Check for visibility of line
   a) If region codes for both endpoints are zero then the line is completely visible. Hence draw the line and go to step 6.
   b) If region codes for endpoints are not zero and the logical ANDing of them is also nonzero then the line is completely invisible, so reject the line and go to step 6.
   c) If region codes for two endpoints do not satisfy the conditions in 4a) and 4b) the line is partially visible.
5. Divide the partially visible line segment in equal parts and repeat steps 3 through 5 for both subdivided line segments until you get completely visible and completely invisible line segments.
6. Stop.

C++ Code for Midpoint Subdivison: -

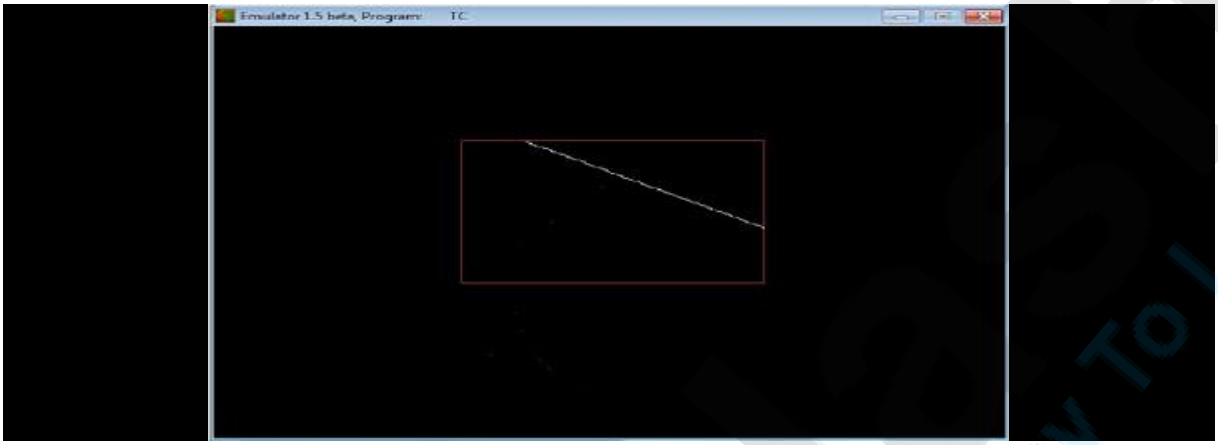
```
//MIDPOINT SUBDIVISION LINE CLIPPING
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<math.h>
#include<graphics.h>
```

```
typedef struct coordinate
{
int x,y;
char code[4];
}PT;
void drawwindow();
void drawline (PT p1,PT p2,int cl);
PT setcode(PT p);

int visibility (PT p1,PT p2);
PT resetendpt (PT p1,PT p2);
main()
{
 int gd=DETECT, gm,v;
 PT p1,p2,ptemp;
 initgraph(&gd,&gm,"C:\\TC\\BGI ");
 cleardevice();
 printf("\n\n\t\tENTER END-POINT 1 (x,y): ");
 scanf("%d,%d",&p1.x,&p1.y);
 printf("\n\n\t\tENTER END-POINT 2 (x,y): ");
 scanf("%d,%d",&p2.x,&p2.y);
 cleardevice();
 drawwindow();
 getch();
 drawline(p1,p2,15);
 getch();
 cleardevice();
 drawwindow();
 midsub(p1,p2);
 getch();
 closegraph();
 return(0);
}


midsub(PT p1,PT p2)
{
 PT mid;
 int v;
 p1=setcode(p1);
 p2=setcode(p2);
 v=visibility(p1,p2);
 switch(v)
  {
  case 0:  /* Line conpletely visible */
    drawline(p1,p2,15);
    break;
  case 1:  /* Line completely invisible */
    break;
  case 2:  /* line partly visible */
    mid.x = p1.x + (p2.x-p1.x)/2;
    mid.y = p1.y + (p2.y-p1.y)/2;
    midsub(p1,mid);
    mid.x = mid.x+1;
    mid.y = mid.y+1;
    midsub(mid,p2);
    break;
  }
}
```

```
void drawwindow()
{
setcolor(RED);
line(150,100,450,100);
line(450,100,450,400);
line(450,400,150,400);
line(150,400,150,100);
}


void drawline (PT p1,PT p2,int cl)
{
setcolor(cl);
line(p1.x,p1.y,p2.x,p2.y);
}


PT setcode(PT p)
{
PT ptemp;
if(p.y<=100)
ptemp.code[0]='1'; /* TOP */
else
ptemp.code[0]='0';
if(p.y>=400)
ptemp.code[1]='1'; /* BOTTOM */
else
ptemp.code[1]='0';
if (p.x>=450)
ptemp.code[2]='1'; /* RIGHT */
else
ptemp.code[2]='0';
if (p.x<=150)      /* LEFT */
ptemp.code[3]='1';
else
ptemp.code[3]='0';
ptemp.x=p.x;
ptemp.y=p.y;
return(ptemp);
}


int visibility (PT p1,PT p2)
{
int i,flag=0;
for(i=0;i<4;i++)
{
if((p1.code[i]!='0')||(p2.code[i]!='0'))
flag=1;
}
if(flag==0)
return(0);
for(i=0;i<4;i++)
{
if((p1.code[i]==p2.code[i]) &&(p1.code[i]=='1'))
flag=0;
}
if(flag==0)
```

```
return(1);
return(2);
}
```

OUTPUT:

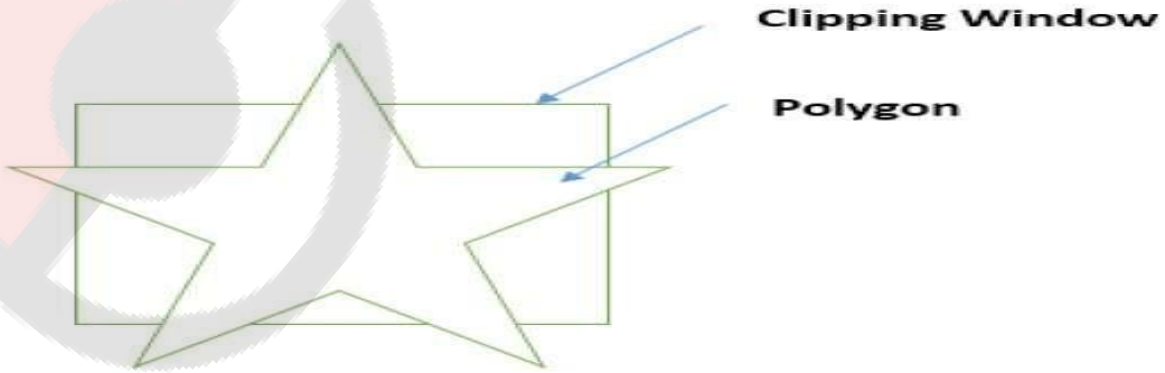


3.5.3) POLYGON CLIPPING:

Sutherland – Hodgeman Polygon Clipping

A polygon can also be clipped by specifying the clipping window. Sutherland Hodgeman polygon clipping algorithm is used for polygon clipping. In this algorithm, all the vertices of the polygon are clipped against each edge of the clipping window.

First the polygon is clipped against the left edge of the polygon window to get new vertices of the polygon. These new vertices are used to clip the polygon against right edge, top edge, bottom edge, of the clipping window as shown in the following figure.



While processing an edge of a polygon with clipping window, an intersection point is found if edge is not completely inside clipping window and the a partial edge from the intersection

point to the outside edge is clipped. The following figures show left, right, top and bottom edge clippings −

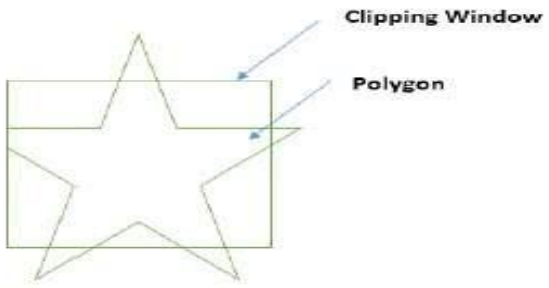


Figure: Clipping Left Edge

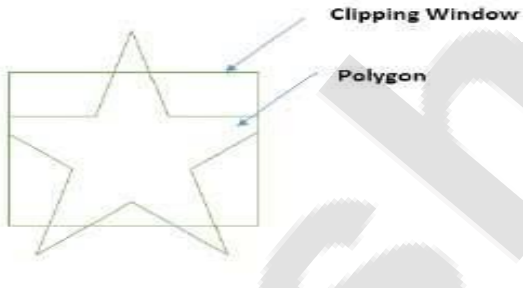


Figure: Clipping Right Edge

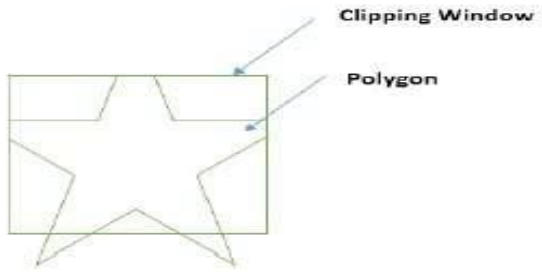


Figure: Clipping Top Edge

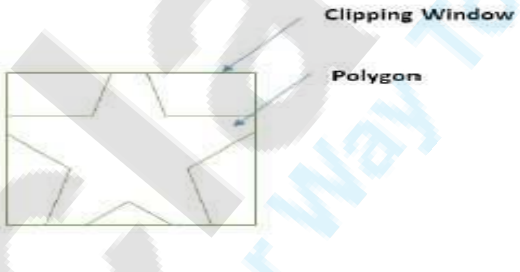


Figure: Clipping Bottom Edge

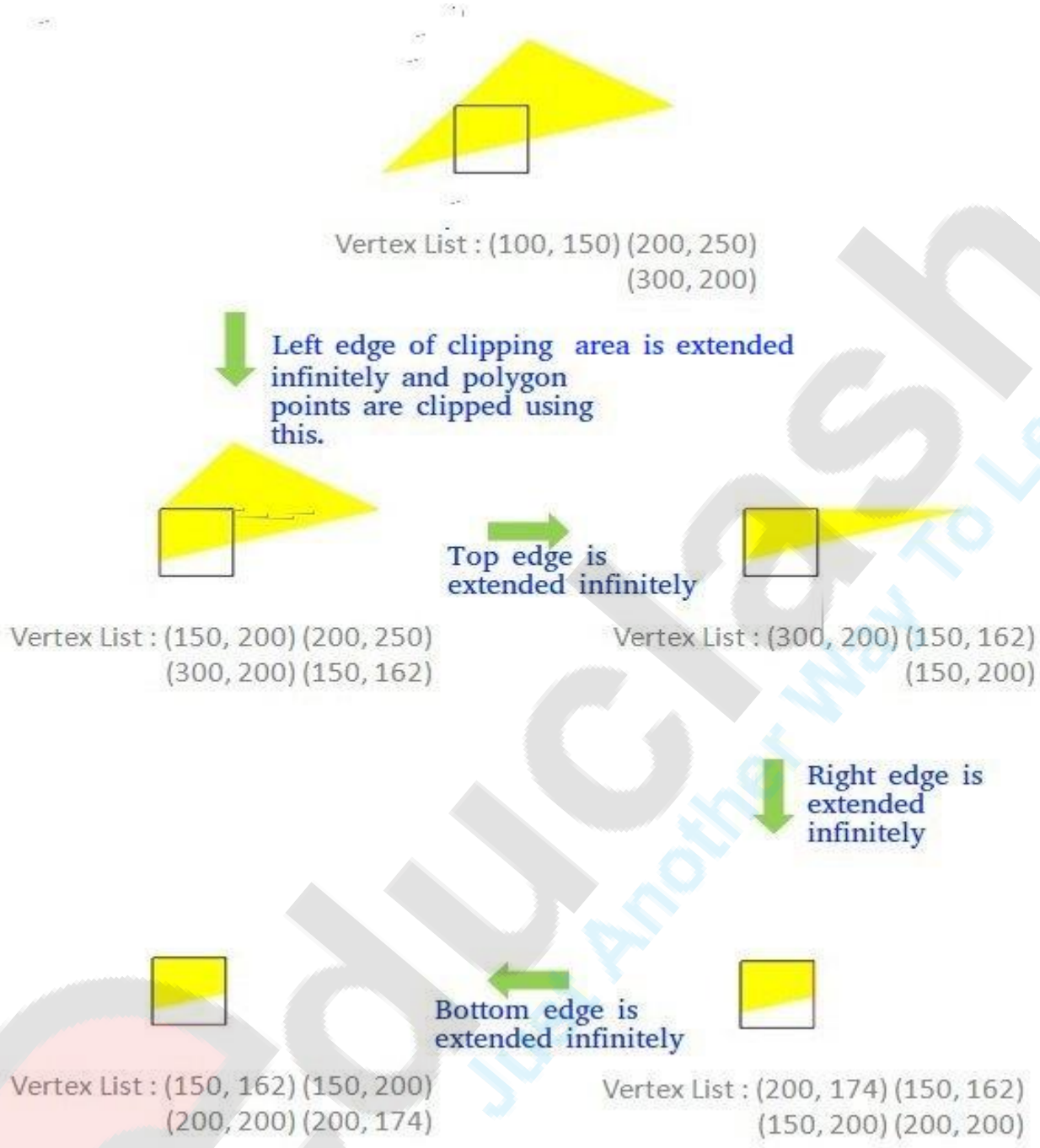
Overview of the algorithm:

```
Consider each edge e of clipping Area and do following:
   a) Clip given polygon against e.
```

How to clip against an edge of clipping area?

The edge (of clipping area) is extended infinitely to create a boundary and all the vertices are clipped using this boundary. The new list of vertices generated is passed to the next edge of the clip polygon in clockwise fashion until all the edges have been used.
There are four possible cases for any given edge of given polygon against current clipping edge e.

1. **Both vertices are inside :** Only the second vertex is added to the output list
2. **First vertex is outside while second one is inside :** Both the point of intersection of the edge with the clip boundary and the second vertex are added to the output list
3. **First vertex is inside while second one is outside :** Only the point of intersection of the edge with the clip boundary is added to the output list
4. **Both vertices are outside :** No vertices are added to the output list



There are two sub-problems that need to be discussed before implementing the algorithm:-

1. **<u>To decide if a point is inside or outside the clipper polygon</u>**
   If the vertices of the clipper polygon are given in clockwise order then all the points lying on the **right side** of the clipper edges are inside that polygon. This can be calculated using :

Given that the line starts from (x_1, y_1) and ends at (x_2, y_2)

$$P = (x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1)$$

if P<0, the point is on the right side of the line

P=0, the point is on the line

P>0, the point is on the left side of the line

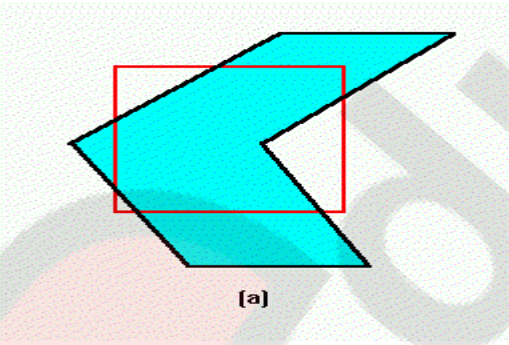
2. **To find the point of intersection of an edge with the clip boundary**

If two points of each line(1,2 & 3,4) are known, then their point of intersection can be calculated using the formula :-

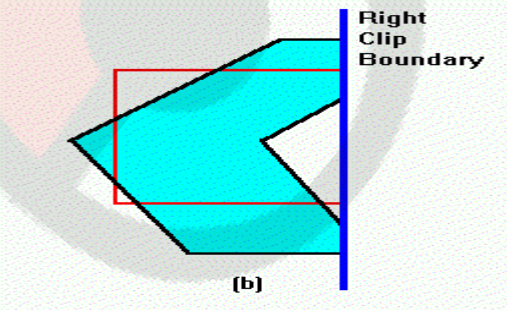
$$(P_x, P_y) = \left(\frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}, \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \right)$$

Step by step example of polygon clipping: -

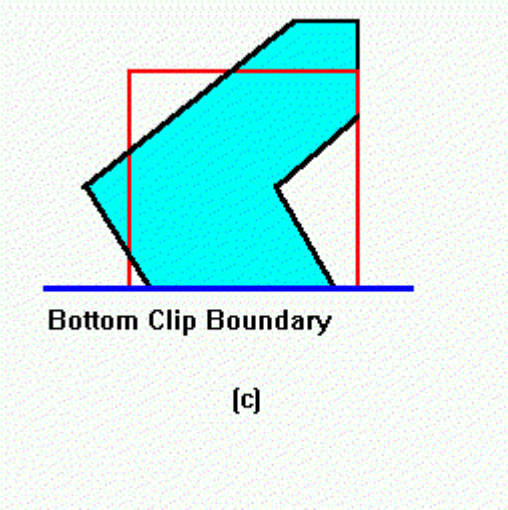
1) **The original polygon and the clip rectangle.**



2) **After clipped by the right clip boundary.**

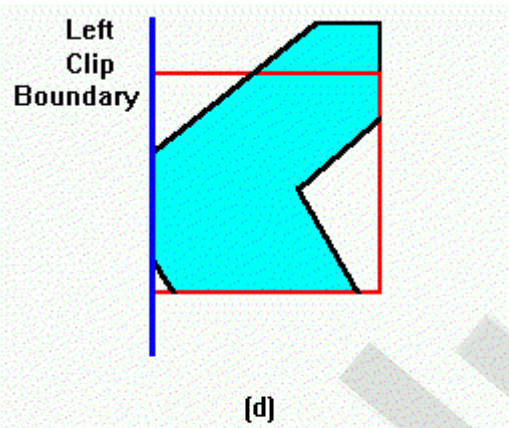


3) After clipped by the right and bottom clip boundaries.



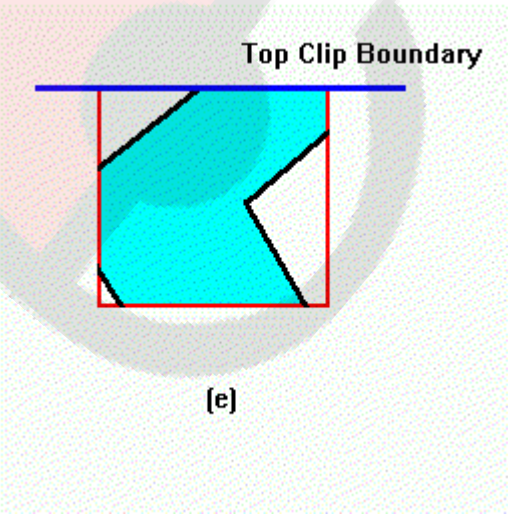
(c)

4) After clipped by the right, bottom, and left clip boundaries.



(d)

5) After clipped by all four boundaries.



(e)

C++ Program for Sutherland – Hodgeman Polygon Clipping

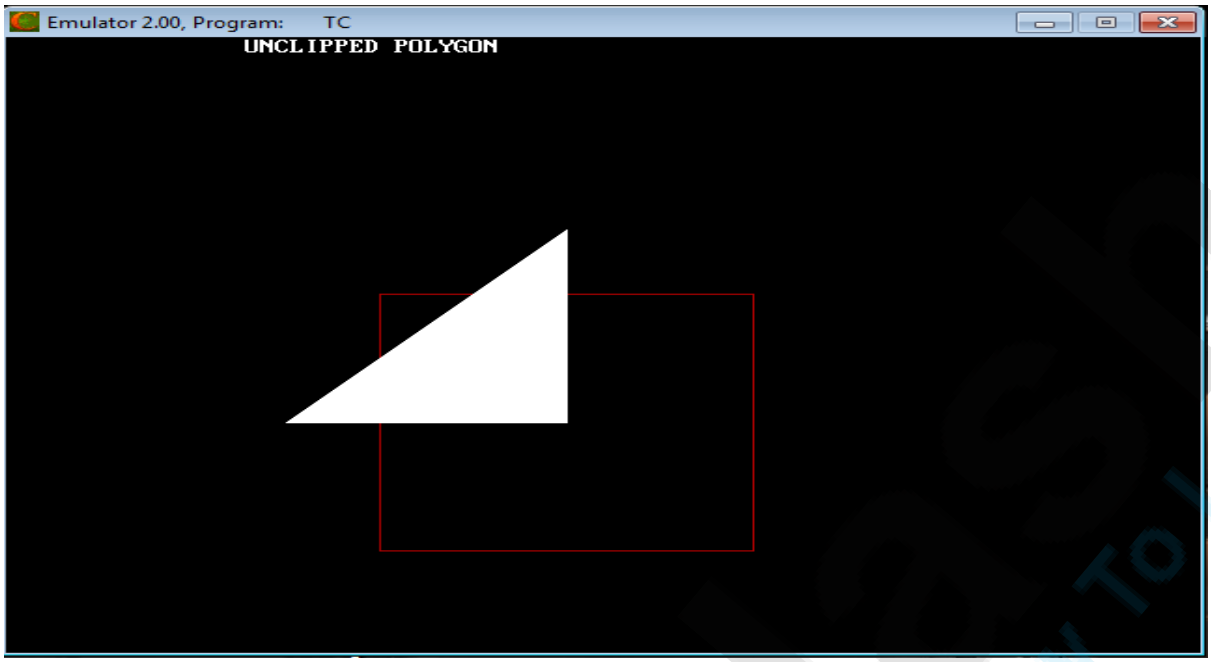
```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#define round(a) ((int)(a+0.5))
int k;
float xmin,ymin,xmax,ymax,arr[20],m;
void clipl(float x1,float y1,float x2,float y2)
{
    if(x2-x1)
        m=(y2-y1)/(x2-x1);
    else
        m=100000;
    if(x1 >= xmin && x2 >= xmin)
    {
        arr[k]=x2;
        arr[k+1]=y2;
        k+=2;
    }
    if(x1 < xmin && x2 >= xmin)
    {
        arr[k]=xmin;
        arr[k+1]=y1+m*(xmin-x1);
        arr[k+2]=x2;
        arr[k+3]=y2;
        k+=4;
    }
    if(x1 >= xmin  && x2 < xmin)
    {
        arr[k]=xmin;
        arr[k+1]=y1+m*(xmin-x1);
        k+=2;
    }
}

void clipt(float x1,float y1,float x2,float y2)
{
    if(y2-y1)
        m=(x2-x1)/(y2-y1);
    else
        m=100000;
    if(y1 <= ymax && y2 <= ymax)
    {
        arr[k]=x2;
        arr[k+1]=y2;
        k+=2;
    }
    if(y1 > ymax && y2 <= ymax)
    {
        arr[k]=x1+m*(ymax-y1);
        arr[k+1]=ymax;
        arr[k+2]=x2;
        arr[k+3]=y2;
        k+=4;
    }
    if(y1 <= ymax  && y2 > ymax)
    {
        arr[k]=x1+m*(ymax-y1);
        arr[k+1]=ymax;
        k+=2;
    }
}

void clipr(float x1,float y1,float x2,float y2)
```

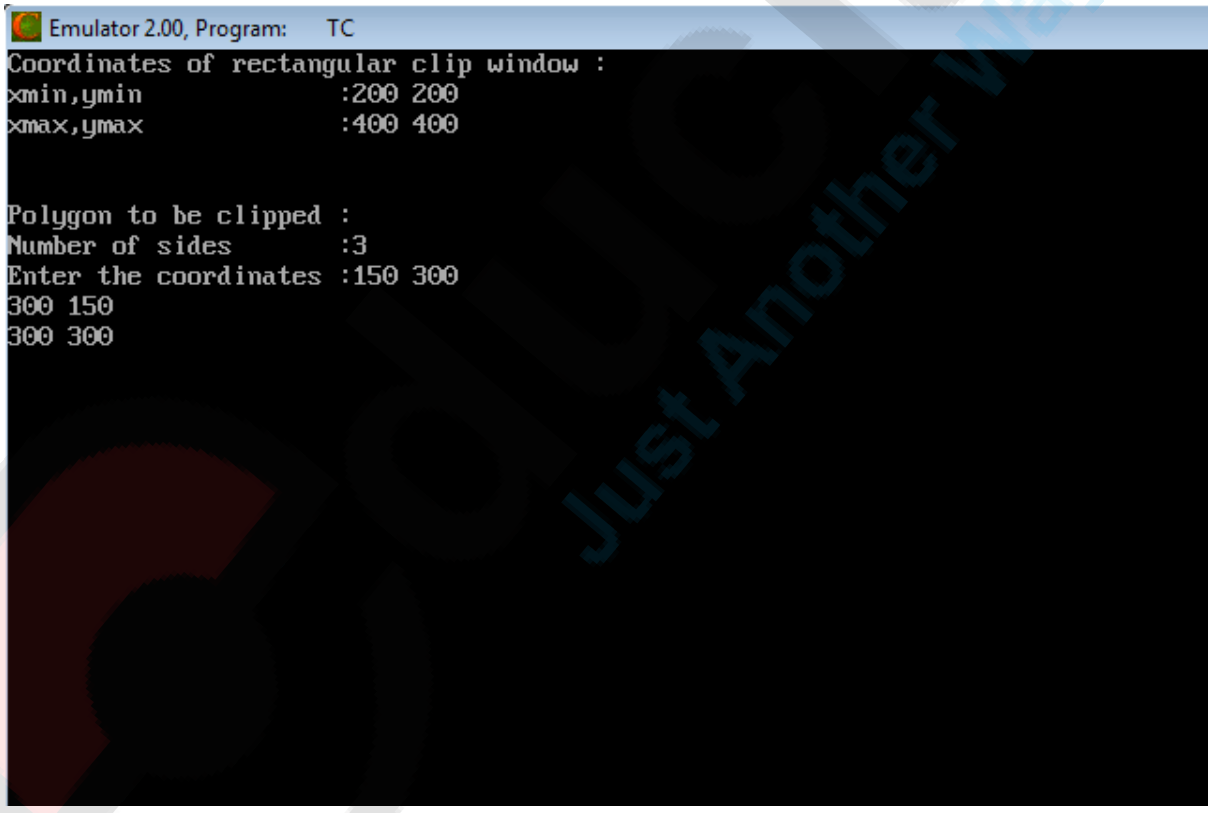
```
{
    if(x2-x1)
        m=(y2-y1)/(x2-x1);
    else
        m=100000;
    if(x1 <= xmax && x2 <= xmax)
    {
        arr[k]=x2;
        arr[k+1]=y2;
        k+=2;
    }
    if(x1 > xmax && x2 <= xmax)
    {
        arr[k]=xmax;
        arr[k+1]=y1+m*(xmax-x1);
        arr[k+2]=x2;
        arr[k+3]=y2;
        k+=4;
    }
    if(x1 <= xmax  && x2 > xmax)
    {
        arr[k]=xmax;
        arr[k+1]=y1+m*(xmax-x1);
        k+=2;
    }
}

void clipb(float x1,float y1,float x2,float y2)
{
    if(y2-y1)
        m=(x2-x1)/(y2-y1);
    else
        m=100000;
    if(y1 >= ymin && y2 >= ymin)
    {
        arr[k]=x2;
        arr[k+1]=y2;
        k+=2;
    }
    if(y1 < ymin && y2 >= ymin)
    {
        arr[k]=x1+m*(ymin-y1);
        arr[k+1]=ymin;
        arr[k+2]=x2;
        arr[k+3]=y2;
        k+=4;
    }
    if(y1 >= ymin  && y2 < ymin)
    {
        arr[k]=x1+m*(ymin-y1);
        arr[k+1]=ymin;
        k+=2;
    }
}

void main()
{
    int gdriver=DETECT,gmode,n,poly[20];
    float xi,yi,xf,yf,polyy[20];
    clrscr();
    cout<<"Coordinates of rectangular clip window :\nxmin,ymin             :";
    cin>>xmin>>ymin;
    cout<<"xmax,ymax             :";
    cin>>xmax>>ymax;
    cout<<"\n\nPolygon to be clipped :\nNumber of sides       :";
    cin>>n;
```

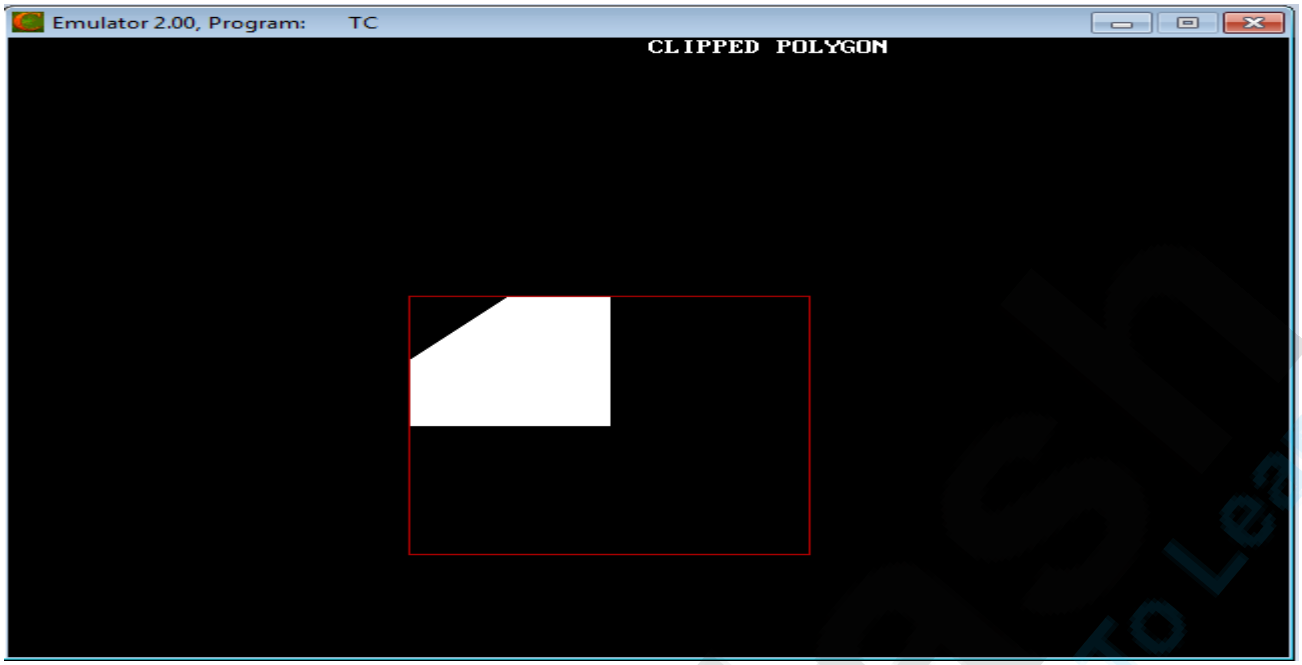
```
    cout<<"Enter the coordinates :";
    for(int i=0;i < 2*n;i++)
              cin>>polyy[i];
    polyy[i]=polyy[0];
    polyy[i+1]=polyy[1];
    for(i=0;i < 2*n+2;i++)
              poly[i]=round(polyy[i]);
    initgraph(&gdriver,&gmode,"C:\\TC\\BGI");
    setcolor(RED);
    rectangle(xmin,ymax,xmax,ymin);
    cout<<"\t\tUNCLIPPED POLYGON";
    setcolor(WHITE);
    fillpoly(n,poly);
       getch();
    cleardevice();
    k=0;
    for(i=0;i < 2*n;i+=2)
              clipl(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
    n=k/2;
    for(i=0;i < k;i++)
              polyy[i]=arr[i];
    polyy[i]=polyy[0];
    polyy[i+1]=polyy[1];
    k=0;
    for(i=0;i < 2*n;i+=2)
              clipt(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
    n=k/2;
    for(i=0;i < k;i++)
              polyy[i]=arr[i];
    polyy[i]=polyy[0];
    polyy[i+1]=polyy[1];
    k=0;
    for(i=0;i < 2*n;i+=2)
              clipr(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
    n=k/2;
    for(i=0;i < k;i++)
              polyy[i]=arr[i];
    polyy[i]=polyy[0];
    polyy[i+1]=polyy[1];
    k=0;
    for(i=0;i < 2*n;i+=2)
              clipb(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
    for(i=0;i < k;i++)
              poly[i]=round(arr[i]);
    if(k)
              fillpoly(k/2,poly);
    setcolor(RED);
    rectangle(xmin,ymax,xmax,ymin);
    cout<<"\tCLIPPED POLYGON";
    getch();
    closegraph();
}
```

Eduklash Just Another Way To Learn

OUTPUT


Emulator 2.00, Program: TC
UNCLIPPED POLYGON


Emulator 2.00, Program: TC
Coordinates of rectangular clip window :
xmin,ymin :200 200
xmax,ymax :400 400
Polygon to be clipped :
Number of sides :3
Enter the coordinates :150 300
300 150
300 300



3.5.4) POLYGON CLIPPING:

Weiler – Atheron Polygon Clipping

The clipping algorithms studied require a convex polygon in context of many applications. e.g. hidden surface removal, the ability to clip to concave polygon is required. A powerful but somewhat more complex clipping algorithm developed by weiler and Atherton meets this requirement. This algorithm defines the Polygon to be clipped as a subject Polygon and the clipping region is the clip Polygon.

The algorithm describes both the subject and the clip polygon by a circular list of vertices. The boundaries of the subject polygon and the clip polygon may or may not intersect. If they intersect, then the intersections occur in pairs. One of the intersections occurs when a subject polygon edge enters the inside of the clip Polygon and one when it leaves. As shown in the figure (n),

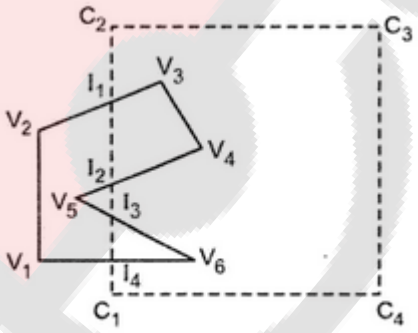


Fig. (n)

there are four intersection vertices I1, I2 ,I3 and I4. In these intersections I1 and 13 are entering intersections, and I2 and I3 are leaving intersections. The clip polygon vertices are marked as C1, C2, C3 and C4.

In this algorithm two separate vertices lists are made one for **clip polygon** and one for **subject polygon** including intersection points. The Table 2 shows these two lists for polygons shown in figure (n).

The algorithm starts at an entering intersection (I1) and follows the subject polygon vertex list in the downward direction (i.e. I1, V3, V4, I1). At the occurrence of leaving intersection the algorithm follows the clip polygon vertex list from the leaving intersection vertex in the downward direction (i.e. I2, I1). At the occurrence of the entering intersection the algorithm follows the subject polygon vertex list from the entering intersection vertex.

For subject polygon		For clip polygon	
	V_1	C_1	
	V_2	I_4	
Start	I_1	I_3	Finish
	V_3	I_2	
	V_4	I_1	Finish
	I_2	C_2	
	V_5	C_3	
Start	I_3	C_4	
	V_6		
	I_4		
	V_1		

Table (2) List of Polygon vertices

This process is repeated until we get the starting vertex. This Process we have to repeat for all remaining entering intersections which are not included in the previous traversing of vertex list. In our example, entering vertex I3 was not included in the first traversing of vertex list, Therefore, we have to go for another vertex traversal from vertex I3.

The above two vertex traversal gives two clipped inside polygons. There are: - **I1, V3, V4, I2, I1 and I3, V6, I4, I3**

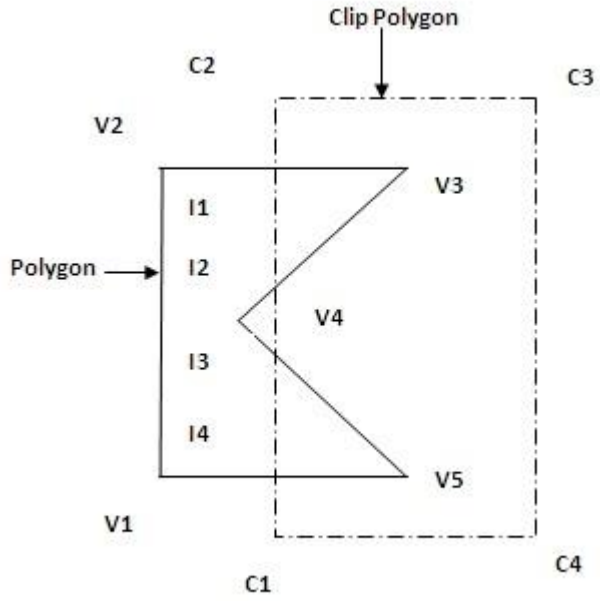
Alternate Explanation :

Given polygon A as the clipping region and polygon B as the subject polygon to be clipped, the algorithm consists of the following steps:

1. List the vertices of the clipping-region polygon A and those of the subject polygon B.
2. Label the listed vertices of subject polygon B as either inside or outside of clipping region A.
3. Find all the polygon intersections and insert them into both lists, linking the lists at the intersections.
4. Generate a list of "inbound" intersections – the intersections where the vector from the intersection to the subsequent vertex of subject polygon B begins inside the clipping region.

5. Follow each intersection clockwise around the linked lists until the start position is found.

This algorithm is used for clipping concave polygons. Here V1, V2, V3, V4, V5 are the vertices of the polygon. C4, C2, C3, C4 are the vertices of the clip polygon and I1, I2, I3, I4 are the intersection points of polygon and clip polygon.



In this algorithm we take a starting vertex like I1 and traverse the polygon like I1, V3, I2. At occurrence of leaving intersection the algorithm follows the clip polygon vertex list from leaving vertex in downward direction. At occurrence of entering intersection the algorithm follows subject polygon vertex list from the entering intersection vertex. This process is repeated till we get starting vertex. This process has to be repeated for all remaining entering intersections which are not included in the previous traversing of vertex list. Since I3 was not included in first traverse, hence, we start the second traversal from I3. Therefore, first traversal gives polygon as: I1, V3, I2, I1 and second traversal gives polygon as: I3, V5, I4, I3.

UNIT 4

BASIC 3D CONCEPTS AND FRACTALS

4.1) 3D Object Representation Methods:

4.1.1) B-Rep:

1. B-Rep stands for Boundary Representation.
2. It is an extension to the wireframe model.
3. B-Rep describes the solid in terms of its surface boundaries: Vertices, edges and faces as shown in figure 35

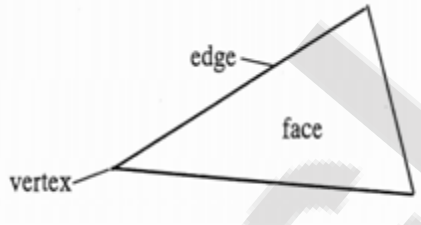


Figure 35

1. It is a method for representing shapes using the limits.
2. A solid is represented as a collection of connected surface elements, the boundary between solid and non-solid.
3. There are 2 types of information in a B – rep topological and geometric.
4. Topological information provides the relationships among vertices, edges and faces similar to that used in a wireframe model.
5. In addition to connectivity, topological information also includes orientation of edges and faces.
6. Geometric information is usually equations of the edges and faces.
7. The B-rep of 2 manifolds that have faces with holes satisfies the generalized Euler's formula:

$$V–E+F–H=2(C–G)V–E+F–H=2(C–G)$$

Where, V = Number of vertices.

E = Number of edges.

F = Number of faces.

H = Number of holes in the faces.

C is the number of separate components (parts).

G is the genus (for a torus G = 1)

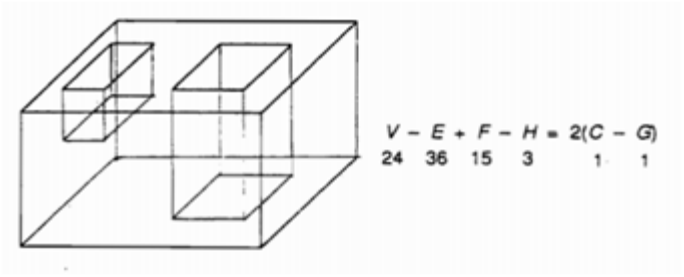


Figure 36

4.1.2) Sweep Representations:

Sweep representations are used to construct 3D object from 2D shape that have some kind of symmetry.

For example, a prism can be generated using a translational sweep and rotational sweeps can be used to create curved surfaces like an ellipsoid or a torus.

In both cases, you start with a cross-section and generate more vertices by applying the appropriate transformation.

More complex objects can be formed by using more complex transformations. Also, we can define a path for a sweep that allows you to create more interesting shapes.

4.1.3) CSG:

- Constructive Solid Geometry (CSG)
  A CSG model assumes that physical objects can be created by combining basic elementary shapes (primitives) through specific rules.

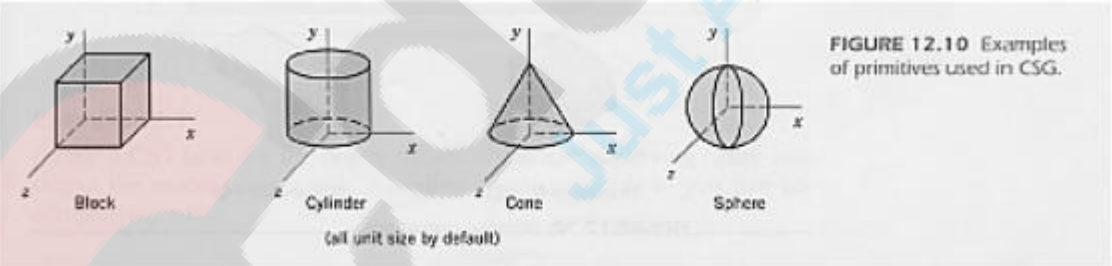


FIGURE 12.10 Examples of primitives used in CSG.

CSG primitives are represented by the intersection of a set of half-spaces, as shown in Figure 12.10.

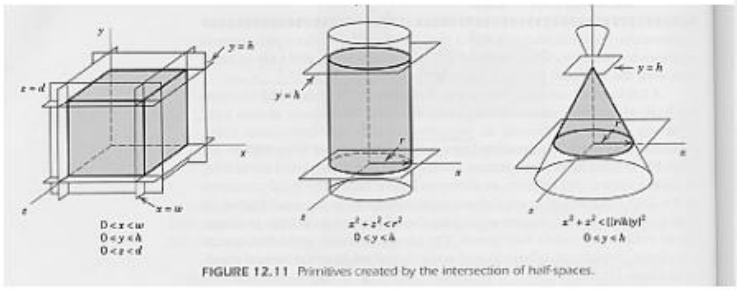


FIGURE 12.11 Primitives created by the intersection of half-spaces.

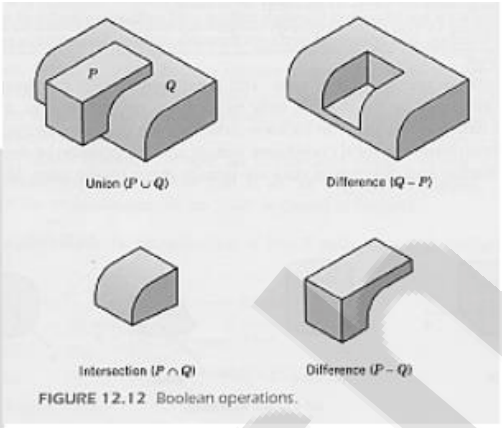


FIGURE 12.12 Boolean operations.

Quadric surfaces are commonly used in CSG because they represent the most commonly used surface in mechanical design produced by the stand operations of milling, turning, rolling. E.g. planar surfaces are obtained through rolling and milling, cylindrical surface through turning, spherical surfaces through cutting done with a ball-end cutting tool. Data structure for the CSG representation is based on the binary tree structure.

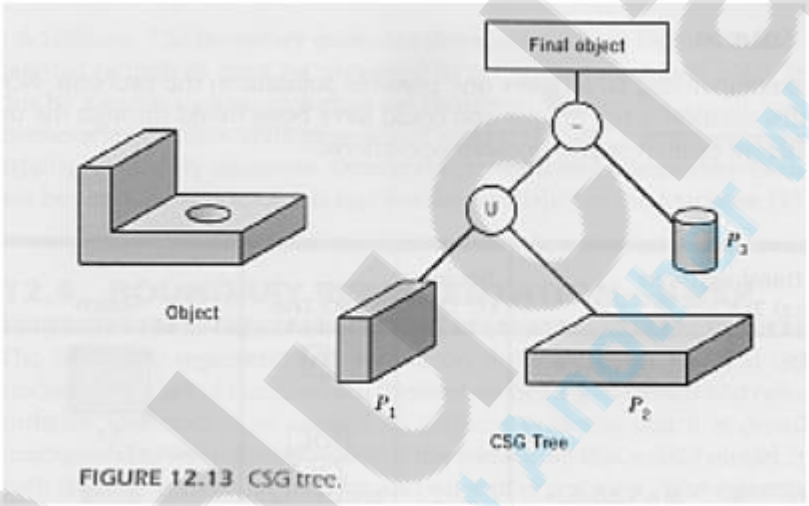


FIGURE 12.13 CSG tree.

CSG example

CSG Table

Primitive No.	Primitive Type	Transformations $S(x,y,z)$ $T(x,y,z)$ $R(x,y,z)$	Boolean (U, D, I)	CSG Tree	Sketch
1	Block	$S(3.0,2.5,.62)$		P_1	
2/3	Block	$S(0.5,2.0,.62)$ $T(2.5,0.5,0.0)$ ‖ $S(0.5,2.0,.62)$ $T(0.0,0.5,0.0)$	D/D	SOL_2 D SOL_1 P_3 D P_1 P_2	
4	Cylinder	$S(r = 0.44, h = 0.62)$ $T(1.5,1.5,0.0)$	D	SOL_3 D SOL_2 P_4	
5	Block	$S(0.56,0.12,0.62)$ $T(1.94,1.44,0.0)$	D	SOL_4 D SOL_3 P_5	
6/7	Cylinder	$S(r = 0.125, h = 0.5)$ $T(0.25,0.0,0.31)$ $R(90.0,0.0,0.0)$ ‖ S(same) $T(2.75,0.0,0.31)$ R(same)	D/D	SOL_5 D SOL_4 P_6 D SOL_1 P_7	

*Default primitives have origin as shown in Figure 12.10, and unit size.

Disadvantages:

1. The way of primitive combinations for the CSG representation is not unique. It has been found through the use of different primitives and Boolean operations.

2. The CSG doesn't specify quantitative values for the new solid (unevaluated model). The new model must be checked through a boundary evaluation routine with will supply quantitative information about its vertices, edges, faces.

3. It shorts of intersection calculation in the form of curve/curve, curve/surface, or surface/surface intersections.

Definition

1. CSG stands for Constructive Solid Geometry.
2. It is based on set of 3D solid primitives and regularized set theoretic operations.
3. Traditional primitives are: Block, cones, sphere, cylinder and torus.
4. Operations: union, intersection, difference + translation and rotation.
5. A complex solid is represented using with a binary tree usually called as CSG tree.
6. CSG tree is shown below in figure 37.

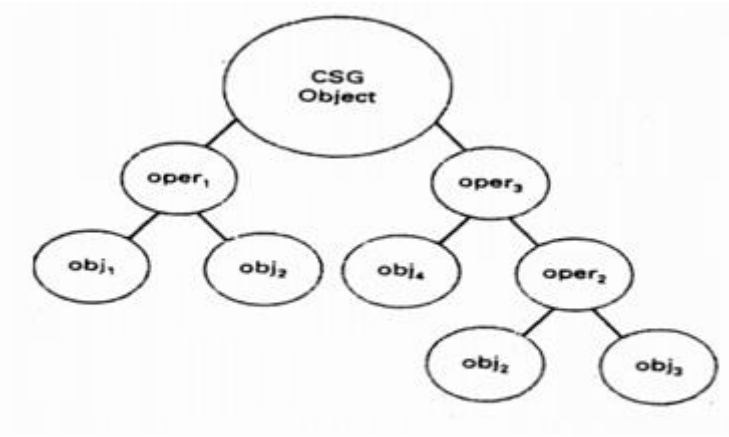


Figure 37

1. Ray casting is a method used for determining boundaries of the resulting object if you start with a boundary representation.
2. Octree representations are designed to make this process easier.

   Example:

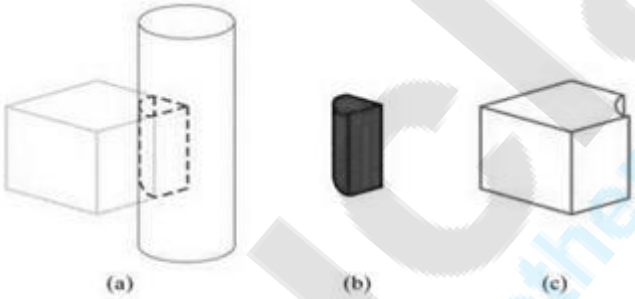


Figure 38

Figure 38 (b) is the intersection of original solid.

Figure 38 (c) is the difference between 2 original solid.

4.1.4) Basic Transformation

In many cases a complex picture can always be treated as a combination of straight line, circles, ellipse etc., and if we are able to generate these basic figures, we can also generate combinations of them. Once we have drawn these pictures, the need arises to transform these pictures.

We are not essentially modifying the pictures, but a picture in the center of the screen needs to be shifted to the top left hand corner, say, or a picture needs to be increased to twice it's size or a picture is to be turned through 90^0. In all these cases, it is possible to view the new picture as really a new one and use algorithms to draw them, but a

better method is, given their present form, try to get their new counter parts by operating on the existing data. This concept is called **transformation**.

The three basic transformations are

(i) Translation

(ii) rotation and

(iii) Scaling.

Translation refers to the shifting of a point to some other place, whose distance with regard to the present point is known. **Rotation** as the name suggests is to rotate a point about an axis. The axis can be any of the coordinates or simply any other specified line also. **Scaling** is the concept of increasing (or decreasing) the size of a picture. (in one or in either directions. When it is done in both directions, the increase or decrease in both directions need not be same) To change the size of the picture, we increase or decrease the distance between the end points of the picture and also change the intermediate points are per requirements.

Translation:

Consider a point P(x_1, y_1) to be translated to another point Q(x_2, y_2). If we know the point value (x_2, y_2) we can directly shift to Q by displaying the pixel (x_2, y_2). On the other hand, suppose we only know that we want to shift by a distance of Tx along x axis and Ty along Y axis. Then obviously the coordinates can be derived by $x_2 = x_1 + Tx$ and $Y_2 = y_1 + T_y$.

Suppose we want to shift a triangle with coordinates at A(20,10), B(30,100) and C(40,70). The shifting to be done by 20 units along x axis and 10 units along y axis. Then the new triangle will be at A^1 (20+20, 10+10) B^1 (30+20, 10+10) C^1 (40+20, 70+10) In the matrix form $[x_2\ y_2\ 1] = [x_1\ y_1\ 1]$

$$* \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & Ty & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A translation in space is described by tx, ty and tz. It is easy to see that this matrix realizes the equations:

x2=x1+tx
y2=y1+ty
z2=z1+tz

Rotation

Suppose we want to rotate a point (x1 y1) clockwise through an angle? about the origin of the coordinate system. Then mathematically we can show that

x2 = x1cos ? + y1sin? and

y2 = x1sin? - y1cos?

These equations become applicable only if the rotation is about the origin.

In the matrix for $[x_2\ y_2\ 1] = [x_1\ y_1\ 1]$

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

Around the Z-axis

$$\begin{bmatrix} cos(v) & -sin(v) & 0 & 0 \\ sin(v) & cos(v) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Around the X-axis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(v) & -sin(v) & 0 \\ 0 & sin(v) & cos(v) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Around the Y-axis

$$\begin{bmatrix} cos(v) & 0 & sin(v) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(v) & 0 & cos(v) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

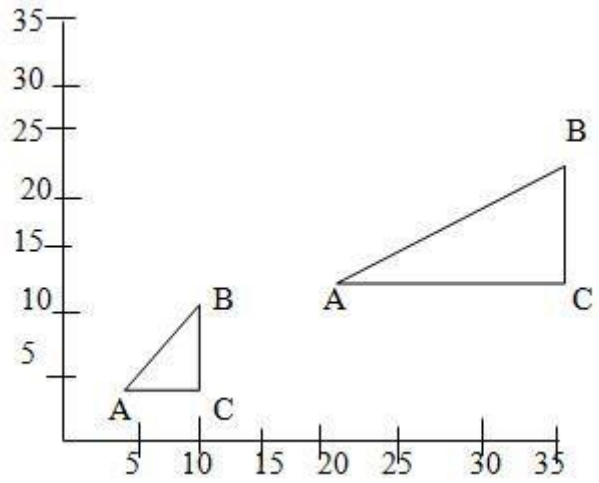
Scaling : Suppose we want the point $(x_1\ y_1)$ to be scaled by a factor sx and by a factor sy along y direction.

Then the new coordinates become : $x_2 = x_1 * sx$ and $y_2 = y_1 * sy$

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & t_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling in space is described by sx, sy and sz. We see that this matrix realizes the following equations:

x2=x1·sx
y2=y1·sy
z2=z1·sz

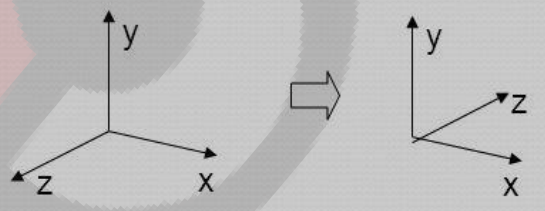


4.1.5) Reflection

A three-dimensional reflection can be performed relative to a selected ***reflection axis*** or with respect to a selected ***reflection plane.*** In general, three-dimensional reflection matrices are set up similarly to those for two dimensions. Reflections relative to a given axis are equivalent to 180o rotations about that axis. Reflections with respect to a plane are equivalent to 180o rotations in four-dimensional space. When the reflection plane is a coordinate plane (either **xy, *xz, or*** yz), we can think of the transformation as a conversion between Left-handed and right-handed systems.

An example of a reflection that converts coordinate specifications from a right-handed system to a left-handed system (or vice versa) is shown in **Fig. 4-3**. This transformation changes the sign of the z coordinates, Leaving the ***x*** and y-coordinate values unchanged. The matrix representation for this reflection of points relative to the **xy** plane is given below

- Reflection Relative to the xy Plane



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

Transformation matrices for inverting **x** and y values are defined similarly, as reflections relative to the yz plane and ***xz*** plane, respectively Reflections about other planes can be obtained as a combination of rotations and coordinate-plane reflections.

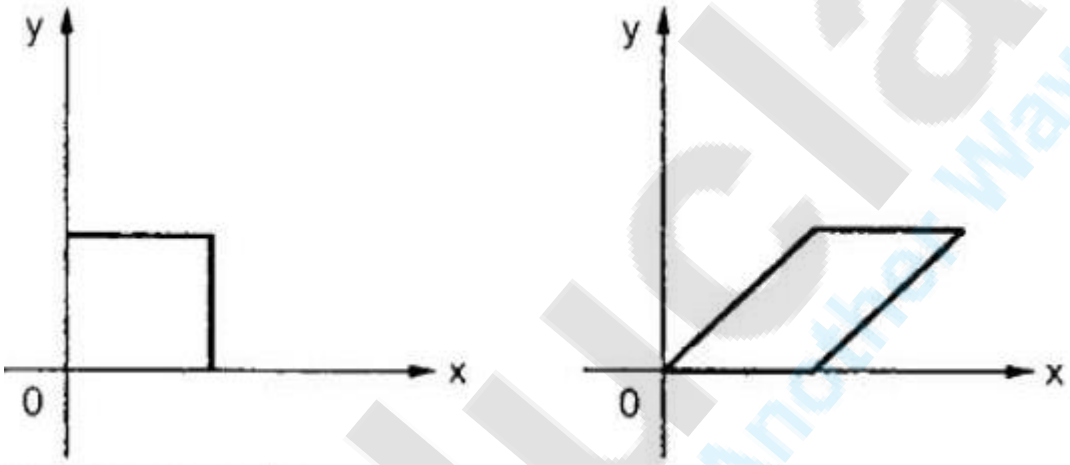
4.1.6) Shears

Shearing:

A transformation that slants the shape of an object is called the shear transformation. There are two shear transformations X-Shear and Y-Shear. One shifts X coordinates values and other shifts Y coordinate values. However; in both the cases only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as Skewing.

X-Shear:

The X-Shear preserves the Y coordinate and changes are made to X coordinates, which causes the vertical lines to tilt right or left as shown in below figure.



(a) Original object (b) Object after x shear

The transformation matrix for X-Shear can be represented as:

$$X_{sh} = \begin{bmatrix} 1 & 0 & 0 \\ shx & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

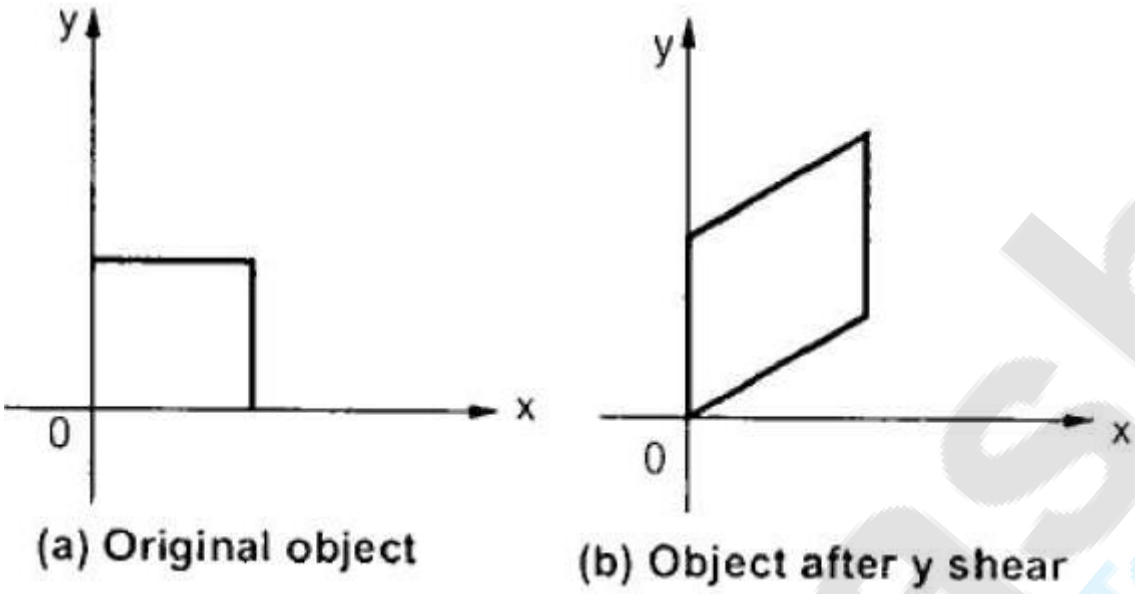
$$X' = X + Sh_X \cdot Y$$

$$Y' = Y$$

Y-Shear:

The Y-Shear preserves the X coordinates and changes the Y coordinates which causes the horizontal lines to transform into lines which slopes up or down as shown in the following

figure.



The Y-Shear can be represented in matrix from as:

$$Y_{sh} \begin{bmatrix} 1 & shy & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Y' = Y + Sh_y \cdot X$$

$$X' = X$$

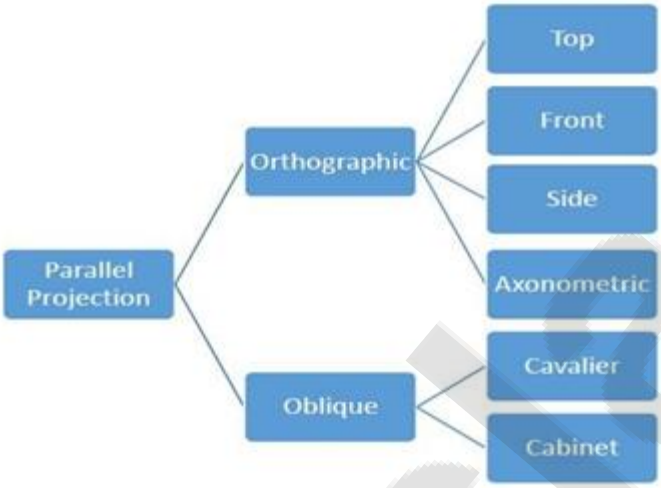
4.2) Projections

Parallel and Perspective

4.2.1) Parallel Projection

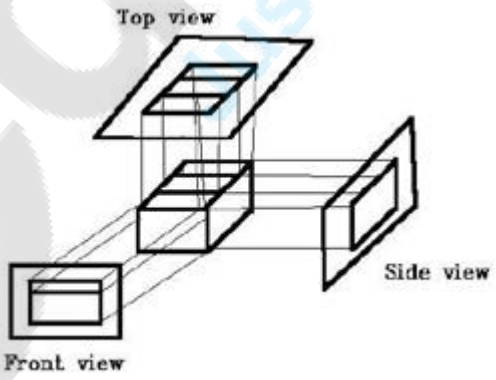
1. Parallel projection discards z-coordinate and parallel lines from each vertex on the object are extended until they intersect the view plane.
2. In parallel projection, we specify a direction of projection instead of center of projection.
3. In parallel projection, the distance from the center of projection to project plane is infinite.

4. In this type of projection, we connect the projected vertices by line segments which correspond to connections on the original object.
5. Parallel projections are less realistic, but they are good for exact measurements.
6. In this type of projections, parallel lines remain parallel and angles are not preserved.
7. Various types of parallel projections are shown in the following hierarchy.



a) Orthographic Projection

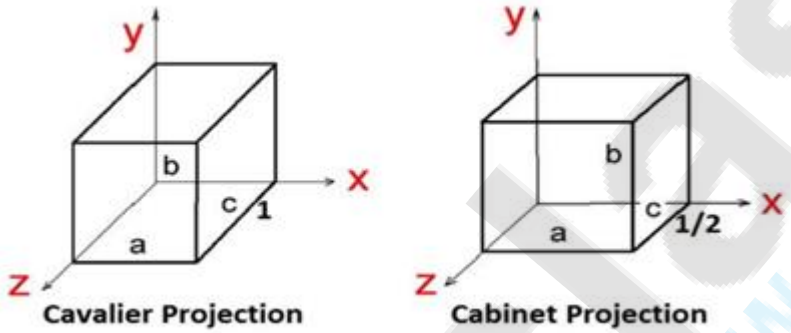
- In orthographic projection the direction of projection is normal to the projection of the plane.
- There are three types of orthographic projections −
    - Front Projection
    - Top Projection
    - Side Projection



b) Oblique Projection

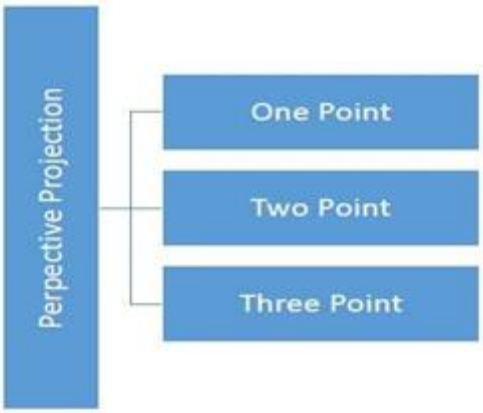
- In orthographic projection, the direction of projection is not normal to the projection of plane.
- In oblique projection, we can view the object better than orthographic projection.
- There are two types of oblique projections − Cavalier and Cabinet.

- The Cavalier projection makes 45° angle with the projection plane.
- The projection of a line perpendicular to the view plane has the same length as the line itself in Cavalier projection.
- In a cavalier projection, the foreshortening factors for all three principal directions are equal.
- The Cabinet projection makes 63.4° angle with the projection plane.
- In Cabinet projection, lines perpendicular to the viewing surface are projected at ½ their actual length.
- Both the projections are shown in the following figure –

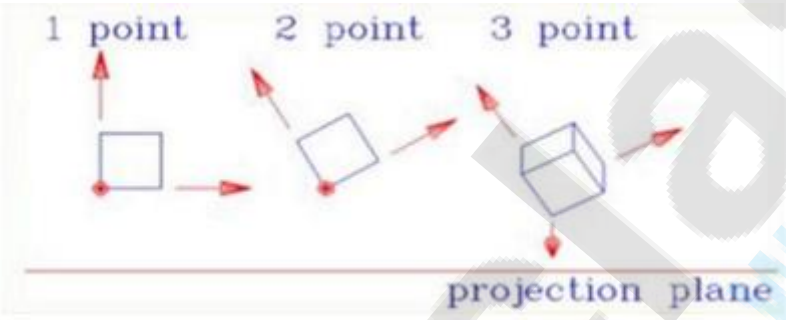


4.2.2) Perspective Projection

1. In perspective projection, the distance from the center of projection to project plane is finite and the size of the object varies inversely with distance which looks more realistic.
2. The distance and angles are not preserved and parallel lines do not remain parallel.
3. Instead, they all converge at a single point called center of projection or projection reference point.
4. There are 3 types of perspective projections which are shown in the following chart.
    - One point perspective projection is simple to draw.
    - Two point perspective projection gives better impression of depth.
    - Three point perspective projection is most difficult to draw.



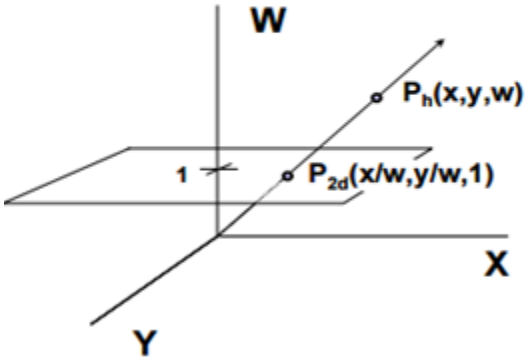
1. The following figure shows all the three types of perspective projection



Matrix for Perspective Projection

- Homogenous coordinates allow to work with the translation as the rotation and scale
    - In order to get a square matrix, a new row is added and a new coordinate w' appears
    - If the last row is [0 0 1] then w' = 1
    - If w'<>1 it is projected over the plane w=1, that's called the homogenous division
    - In the case of 3D, we work with 4 components and the 3D point we get is [x/w, y/w, z/w, 1]

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



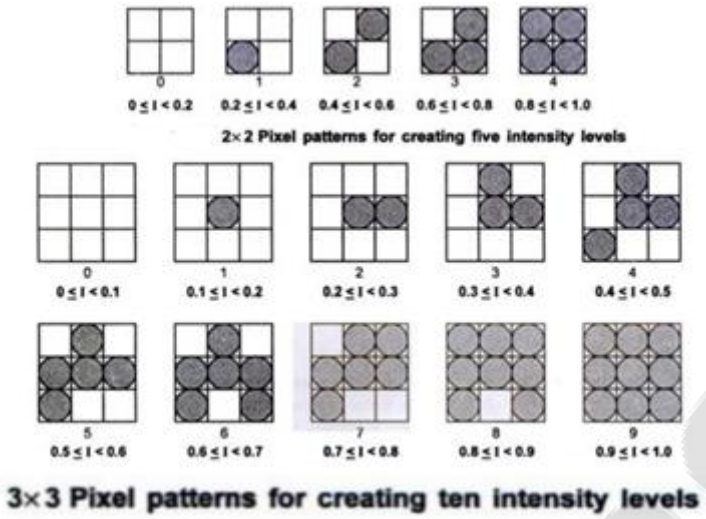
- Simple projection matrices
    - A LEFT - HANDED system is used
    - CP at the origin of coordinates and PP perpendicular to Z at a distance of d
- Simple perspective projection

4.3) Halftone and Dithering

4.3.1) Half toning

1. Many displays and hardcopy devices are bi-level
2. They can only produce two intensity levels.
3. In such displays or hardcopy devices we can create an apparent increase in the number of available intensity value.
4. When we view a very small area from a sufficient large viewing distance, our eyes average fine details within the small area and record only the overall intensity of the area.
5. The phenomenon of apparent increase in the number of available intensities by considering combine intensity of multiple pixels is known as half toning.
6. The half toning is commonly used in printing black and white photographs in newspapers, magazines and books.
7. The pictures produced by half toning process are called halftones.
8. In computer graphics, halftone reproductions are approximated using rectangular pixel regions, say 2*2 pixels or 3*3 pixels.
9. These regions are called halftone patterns or pixel patterns.

10. Figure shows the halftone pattern to create number of intensity levels.



3×3 Pixel patterns for creating ten intensity levels

4.3.2) Dithering Techniques

- Dithering refers to techniques for approximating halftones without reducing resolution, as pixel grid patterns do.
- The term dithering is also applied to halftone approximation method using pixel grid, and something it is used to refer to color halftone approximations only.
- Random values added to pixel intensities to break up contours are often referred as dither noise.
- Number of methods is used to generate intensity variations.
- Ordered dither methods generate intensity variations with a one-to-one mapping of points in a scene to the display pixel.
- To obtain n2n2 intensity levels, it is necessary to set up an n*n dither matrix DnDn whose elements are distinct positive integers in the range of 0 to n2−1n2−1.
- For example it is possible to generate four intensity levels with

$$D_2 = \begin{matrix} 3 & 1 \\ 0 & 2 \end{matrix}$$ and it is possible to generate nine intensity levels with

$$D_3 = \begin{matrix} 7 & 2 & 6 \\ 4 & 0 & 1 \\ 3 & 8 & 5 \end{matrix}$$

- The matrix element for D_2 and D_3 are in the same order as the pixel mask for setting up 2*2 and 3*3 pixel grid respectively.
- For bi-level system, we have to determine display intensity values by comparing input intensities to the matrix elements.
- Each input intensity is first scaled to the range 0< I < n2n2.
- If the intensity I is to be applied to screen position (x, y), we have to calculate numbers for the either matrix as

i=(xmodn)+1 j=(ymodn)+1i=(xmodn)+1 j=(ymodn)+1

- If I>Dn(i,j)I>Dn(i,j) the pixel at position (x, y) is turned on; otherwise the pixel is not turned on.
- Typically the number of intensity levels is taken to be a multiple of 2.
- High order dither matrices can be obtained from lower order matrices with the recurrence relation.

$$D_n = \begin{matrix} 4D_{n/2} + D_2(1,1)u_{n/2} & 4D_{n/2} + D_2(1,2)u_{n/2} \\ 4D_{n/2} + D_2(2,1)u_{n/2} & 4D_{n/2} + D_2(2,\ 2)u_{n/2} \end{matrix}$$

- Another method for mapping a picture with m*n points to a display area with m*n pixels is error diffusion.
- Here, the error between an input intensity value and the displayed pixel intensity level at a given position is dispersed, or diffused to pixel position to the right and below the current pixel position.

4.4) Fractals and self –similarity

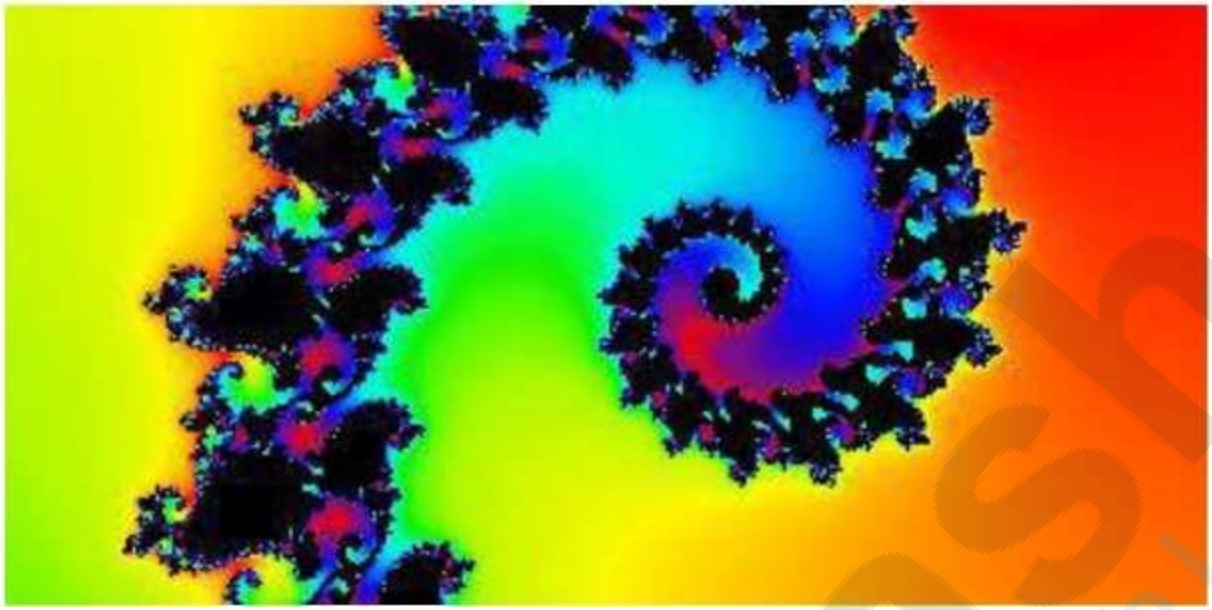
Fractals:

What are Fractals?

Fractals are very complex pictures generated by a computer from a single formula. They are created using iterations. This means one formula is repeated with slightly different values over and over again, taking into account the results from the previous iteration.

Fractals are used in many areas such as −

- **Astronomy** − For analyzing galaxies, rings of Saturn, etc.
- **Biology/Chemistry** − For depicting bacteria cultures, Chemical reactions, human anatomy, molecules, plants,
- **Others** − For depicting clouds, coastline and borderlines, data compression, diffusion, economy, fractal art, fractal music, landscapes, special effect, etc.



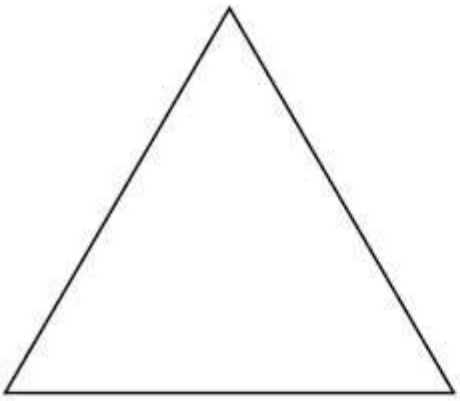
Generation of Fractals

Fractals can be generated by repeating the same shape over and over again as shown in the following figure. In figure (a) shows an equilateral triangle. In figure (b), we can see that the triangle is repeated to create a star-like shape. In figure (c), we can see that the star shape in figure (b) is repeated again and again to create a new shape.

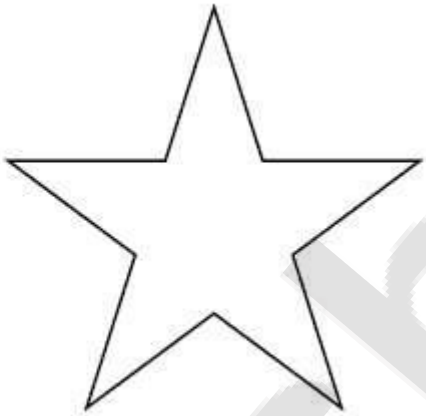
We can do unlimited number of iteration to create a desired shape. In programming terms, recursion is used to create such shapes.

Geometric Fractals

Geometric fractals deal with shapes found in nature that have non-integer or fractal dimensions. To geometrically construct a deterministic (nonrandom) self-similar fractal, we start with a given geometric shape, called the **initiator**. Subparts of the initiator are then replaced with a pattern, called the **generator**.



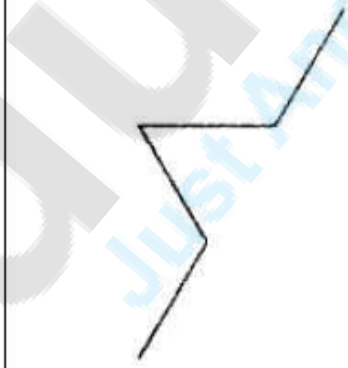
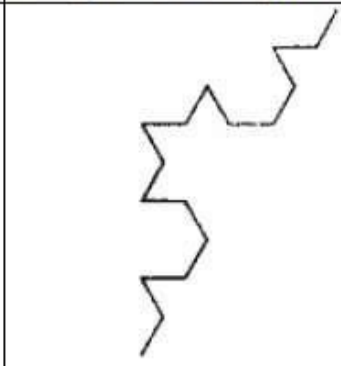
Initiator



Generator

As an example, if we use the initiator and generator shown in the above figure, we can construct good pattern by repeating it. Each straight-line segment in the initiator is replaced with four equal-length line segments at each step. The scaling factor is 1/3, so the fractal dimension is D = ln 4/ln 3 ≈ 1.2619.

Also, the length of each line segment in the initiator increases by a factor of 4/3 at each step, so that the length of the fractal curve tends to infinity as more detail is added to the curve as shown in the following figure –

Segment Length = 1	Segment Length = 1/3	Segment Length = 1/9
		
Length = 1	Length = 4/3	Length = 16/9

4.4.1) Koch Curves/ Snow flake :

1. The Koch snowflake (also known as the Koch curve, star, or island) is a mathematical curve and one of the earliest fractal curves to have been described.
2. A Koch curve is a fractal generated by a replacement rule. This rule is, at each step, to replace the middle 131/3 of each line segment with two sides of a right triangle having sides of length equal to the replaced segment.

3. This quantity increases without bound; hen
4. ce the Koch curve has infinite length.
5. However, the curve still bounds a finite area.
6. We can prove this by noting that in each step, we add an amount of area equal to the area of all the equilateral triangles created.

Construction:

1. The Koch snowflake can be constructed by starting with an equilateral triangle, then recursively altering each line segment as follows:
    - Divide the line segment into three segments of equal length.
    - Draw an equilateral triangle that has the middle segment from step 1 as its base and points outward.

- Remove the line segment that is the base of the triangle from step 2.

- After one iteration of this process, the resulting shape is the outline of a hexagram.

1. The Koch snowflake is the limit approached as the above steps are followed over and over again.
2. The Koch curve originally described by Koch is constructed with only one of the three sides of the original triangle.
3. In other words, three Koch curves make a Koch snowflake.

Koch Curve or Koch Snowflake

What is Koch Curve?

The Koch snowflake (also known as the Koch curve, Koch star, or Koch island) is a mathematical curve and one of the earliest fractal curves to have been described. It is based

on the Koch curve, which appeared in a 1904 paper titled “On a continuous curve without tangents, constructible from elementary geometry” by the Swedish mathematician Helge von Koch.

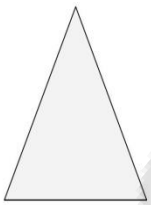
The progression for the area of the snowflake converges to 8/5 times the area of the original triangle, while the progression for the snowflake’s perimeter diverges to infinity. Consequently, the snowflake has a finite area bounded by an infinitely long line.

Construction

Step1:

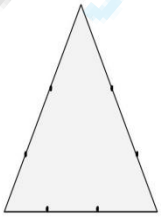
Draw an equilateral triangle. You can draw it with a compass or protractor, or just eyeball it if you don’t want to spend too much time drawing the snowflake.

 It’s best if the length of the sides are divisible by 3, because of the nature of this fractal. This will become clear in the next few steps.

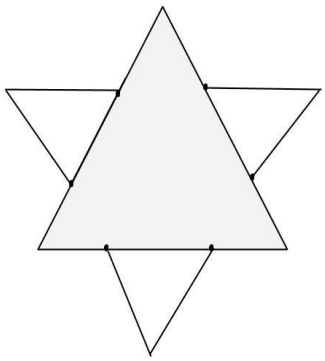


Step2:

Divide each side in three equal parts. This is why it is handy to have the sides divisible by three.

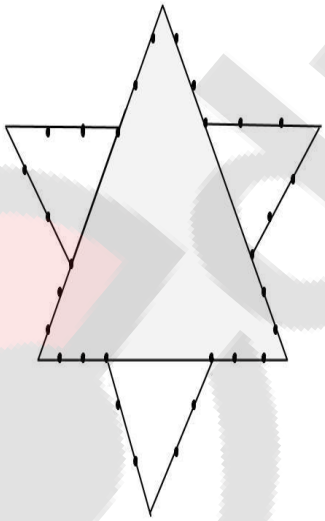


Step3: Draw an equilateral triangle on each middle part. Measure the length of the middle third to know the length of the sides of these new triangles.



Step4:

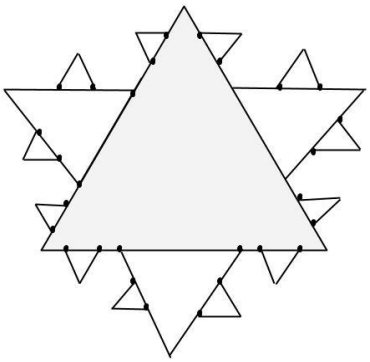
Divide each outer side into thirds. You can see the 2nd generation of triangles covers a bit of the first. These three line segments shouldn't be parted in three.



Step5:

Draw an equilateral triangle on each middle part.

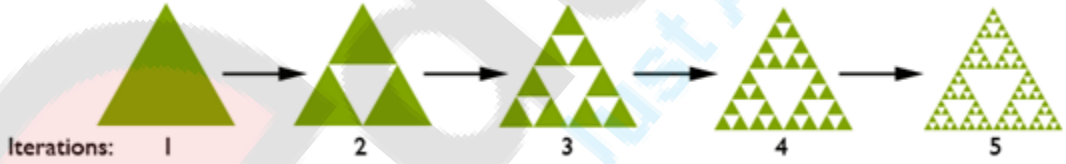
 Note how you draw each next generation of parts that are one 3rd of the mast one.



4.4.2)Sierpinski Triangle(Gasket)

The Sierpinski gasket is a fractal and a very basic fractal of self-similar sets, a mathematically generated pattern with similar patterns. The Sierpisnki gasket is typically generated using triangles, but you can render any shape you want in the space, and it's an example of an iterated function system (IFS).

The Sierpinski gasket can be generated in many different ways. You can create a triangle, and cut out the parts that that's not part of the Sierpinski gasket, or draw dots that after many plots, fill out the entire thing.



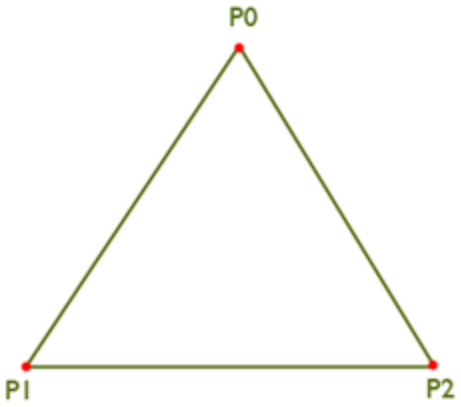
Programming your first fractal using dots

Implementing the Sierpinski gasket using dots is actually very simple. All you need to know is how to divide! You draw dots many at given positions, using the previous position as the

input to the next position.

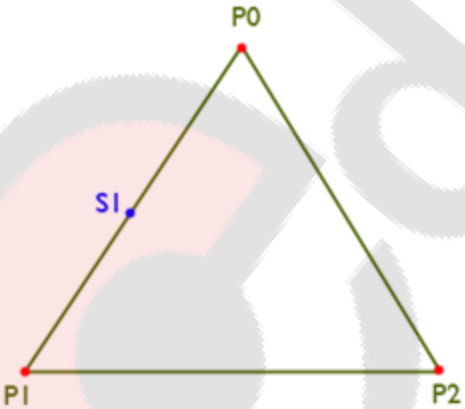
dot position = f(x)
dot position = f(f(x))
dot position = f(f(f(x)))
dot position = f(f(f(f(x))))

So, each point S_k is based in the previous point S_{k-1}.
Next we need to know what the function f(x) does, except for returning the coordinate of a point inside the Sierpinski gasket.
1. First of all, you need to select three points that form a triangle of any sort. These three points is defining the edges of the triangle:



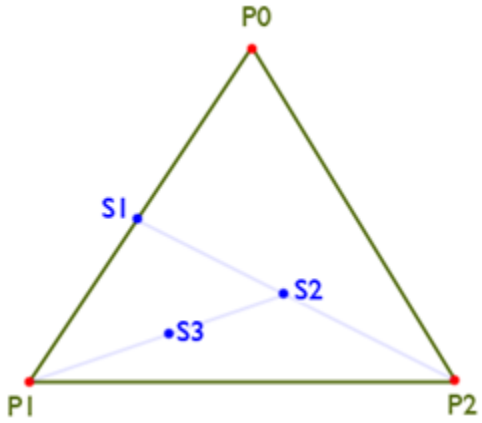
Here we define point P0, P1 and P2 in a 2d coordinate system.

2. Next, use a random function to select any of the three points as a starting point. Let's name this P_{start}.
3. Create the next point S_k as the midpoint between a random edge point P and the previous point S_{k-1}:



4. Draw S_k.
5. Jump to step 3

After a few iterations, this is what you will see:



S1 is the midpoint between P0 and P1, S2 is the midpoint between S1 and P2, and S3 is the midpoint between S2 and P1.

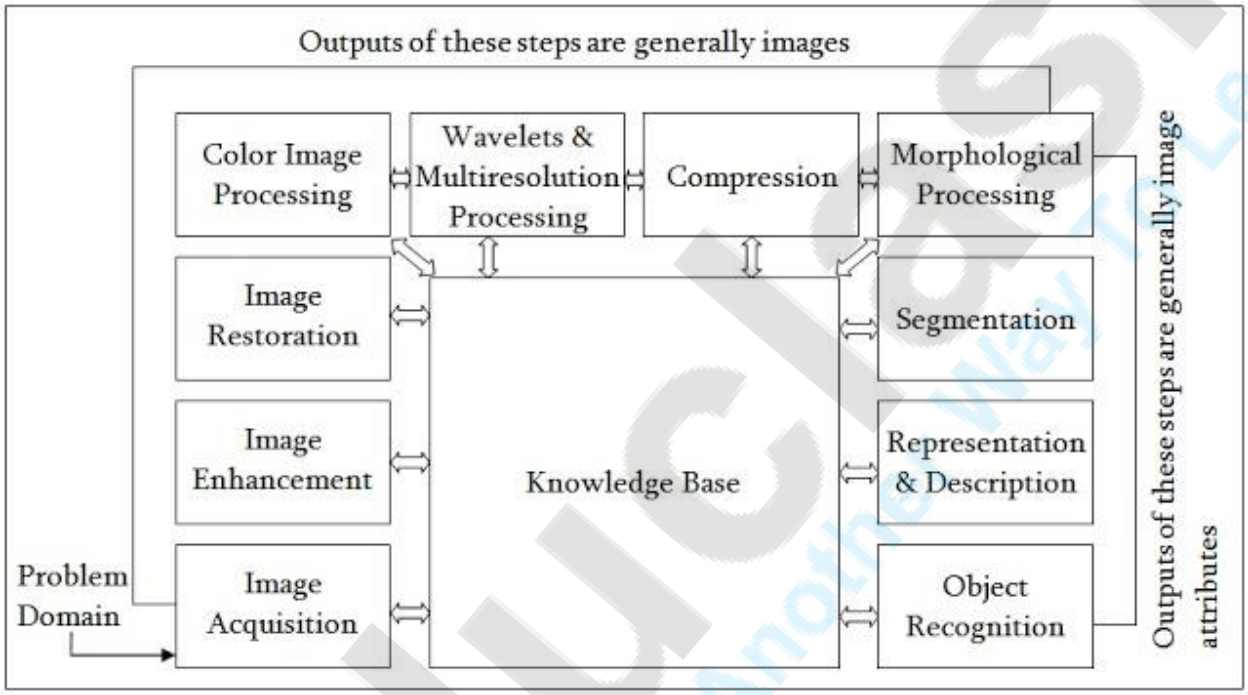
Eduklash Just Another Way To Learn

UNIT 5

INTRODUCTION TO IMAGE PROCESSING

5.1) Fundamental Steps in Digital Image Processing

There are some fundamental steps but as they are fundamental, all these steps may have sub-steps. The fundamental steps are described below with a neat diagram.



1. Image Acquisition:

This is the first step or process of the fundamental steps of digital image processing. Image acquisition could be as simple as being given an image that is already in digital form. Generally, the image acquisition stage involves pre-processing, such as scaling etc.

2. Image Enhancement:

Image enhancement is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interest in an image. Such as, changing brightness & contrast etc.

3. Image Restoration:

Image restoration is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is objective, in the

sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation.

4. Color Image Processing:

Color image processing is an area that has been gaining its importance because of the significant increase in the use of digital images over the Internet. This may include color modeling and processing in a digital domain etc.

5. Wavelets and Multi-Resolution Processing:

Wavelets are the foundation for representing images in various degrees of resolution. Images subdivision successively into smaller regions for data compression and for pyramidal representation.

6. Compression:

Compression deals with techniques for reducing the storage required to save an image or the bandwidth to transmit it. Particularly in the uses of internet it is very much necessary to compress data.

7. Morphological Processing:

Morphological processing deals with tools for extracting image components that are useful in the representation and description of shape.

8. Segmentation:

Segmentation procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually.

9. Representation and Description:

Representation and description almost always follow the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region or all the points in the region itself. Choosing a representation is only part of the solution for transforming raw data into a form suitable for subsequent computer processing. Description deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another.

10. Object recognition:

Recognition is the process that assigns a label, such as, "vehicle" to an object based on its descriptors.

11. Knowledge Base:

Knowledge may be as simple as detailing regions of an image where the information of interest is known to be located, thus limiting the search that has to be conducted in seeking that information. The knowledge base also can be quite complex, such as an interrelated list of all major possible defects in a materials inspection problem or an image database containing high-resolution satellite images of a region in connection with change-detection applications.

5.2) Components of an Image Processing System

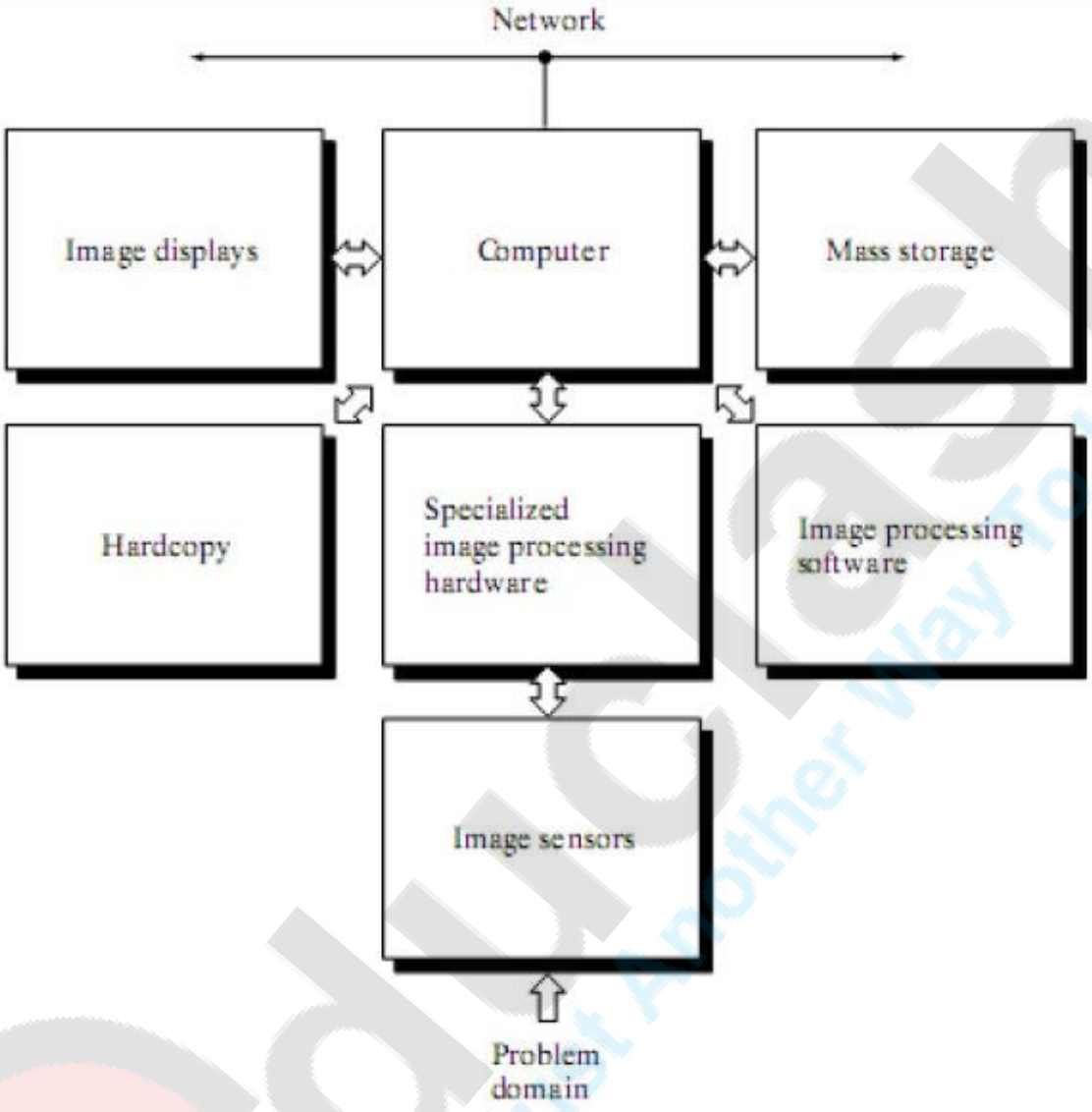
As recently as the mid-1980s, numerous models of image processing systems being sold throughout the world were rather substantial peripheral devices that attached to equally substantial host computers. Late in the 1980s and early in the 1990s, the market shifted to image processing hardware in the form of single boards designed to be compatible with industry standard buses and to fit into engineering workstation cabinets and personal computers. In addition to lowering costs, this market shift also served as a catalyst for a significant number of new companies whose specialty is the development of software written specifically for image processing.

Although large-scale image processing systems still are being sold for massive imaging applications, such as processing of satellite images, the trend continues toward miniaturizing and blending of general-purpose small computers with specialized image processing hardware. Figure 3 shows the basic components comprising a typical general-purpose system used for digital image processing. The function of each component is discussed in the following paragraphs, starting with image sensing.

With reference to sensing, two elements are required to acquire digital images. The first is a physical device that is sensitive to the energy radiated by the object we wish to image. The second, called a digitizer, is a device for converting the output of the physical sensing device into digital form. For instance, in a digital video camera, the sensors produce an electrical output proportional to light intensity. The digitizer converts these outputs to digital data.

Specialized image processing hardware usually consists of the digitizer just mentioned, plus hardware that performs other primitive operations, such as an arithmetic logic unit (ALU), which performs arithmetic and logical operations in parallel on entire images. One example of how an ALU is used is in averaging images as quickly as they are digitized, for the purpose of noise reduction. This type of hardware sometimes is called a front-end subsystem, and its most distinguishing characteristic is speed. In other words, this unit performs functions that require fast data throughputs (e.g., digitizing and averaging video images at 30 framess) that

the typical
main computer cannot handle.



The computer in an image processing system is a general-purpose computer and can range from
a PC to a supercomputer. In dedicated applications, some times specially designed computers are
used to achieve a required level of performance, but our interest here is on general-purpose image processing systems. In these systems, almost any well-equipped PC-type machine is suitable for offline image processing tasks.

Software for image processing consists of specialized modules that perform specific tasks. A well-designed package also includes the capability for the user to write code that, as a minimum,
utilizes the specialized modules. More sophisticated software packages allow the

integration of those modules and general-purpose software commands from at least one computer language.

Mass storage capability is a must in image processing applications. An image of size 1024*1024 pixels, in which the intensity of each pixel is an 8-bit quantity, requires one megabyte of storage space if the image is not compressed. When dealing with thousands, or even millions, of images, providing adequate storage in an image processing system can be a challenge. Digital storage for image processing applications falls into three principal categories: (1) short-term storage for use during processing, (2) on-line storage for relatively fast re-call, and (3) archival storage, characterized by infrequent access. Storage is measured in bytes (eight bits), Kbytes (one thousand bytes), Mbytes (one million bytes), Gbytes (meaning giga, or one billion, bytes), and Tbytes (meaning tera, or one trillion, bytes). One method of providing short-term storage is computer memory. Another is by specialized boards, called frame buffers, that store one or more images and can be accessed rapidly, usually at video rates (e.g., at 30 complete images per second). The latter method allows virtually instantaneous image zoom, as well as scroll (vertical shifts) and pan (horizontal shifts). Frame buffers usually are housed in the specialized image processing hardware unit shown in Fig.3.Online storage generally takes the form of magnetic disks or optical-media storage. The key factor characterizing on-line storage is frequent access to the stored data. Finally, archival storage is characterized by massive storage requirements but infrequent need for access. Magnetic tapes and optical disks housed in “jukeboxes” are the usual media for archival applications.

Image displays in use today are mainly color (preferably flat screen) TV monitors. Monitors are driven by the outputs of image and graphics display cards that are an integral part of the computer system. Seldom are there requirements for image display applications that cannot be met by display cards available commercially as part of the computer system. In some cases, it is necessary to have stereo displays, and these are implemented in the form of headgear containing two small displays embedded in goggles worn by the user.

Hardcopy devices for recording images include laser printers, film cameras, heat-sensitive devices, inkjet units, and digital units, such as optical and CD-ROM disks. Film provides the highest possible resolution, but paper is the obvious medium of choice for written material. For presentations, images are displayed on film transparencies or in a digital medium if image projection equipment is used. The latter approach is gaining acceptance as the standard for image presentations.

Networking is almost a default function in any computer system in use today. Because of the large amount of data inherent in image processing applications, the key consideration in image transmission is bandwidth. In dedicated networks, this typically is not a problem, but communications with remote sites via the Internet are not always as efficient. Fortunately,

this
situation is improving quickly as a result of optical fiber and other broadband technologies.

5.3) Basic Concepts in Sampling and Quantization

Image Sampling and Quantization:

The output of most sensors is a continuous voltage waveform whose amplitude and spatial behavior are related to the physical phenomenon being sensed. To create a digital image, we
need to convert the continuous sensed data into digital form. This involves two processes: sampling and quantization.

Basic Concepts in Sampling and Quantization:

The basic idea behind sampling and quantization is illustrated in Fig.6.1. Figure 6.1(a) shows a
continuous image, f(x, y), that we want to convert to digital form. An image may 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. Digitizing the coordinate values is called sampling. Digitizing the amplitude values is called quantization.

The one-dimensional function shown in Fig.6.1 (b) is a plot of amplitude (gray level) values of
the continuous image along the line segment AB in Fig. 6.1(a).The random variations are due to
image noise. To sample this function, we take equally spaced samples along line AB, as shown
in Fig.6.1 (c).The location of each sample is given by a vertical tick mark in the bottom part of
the figure. The samples are shown as small white squares superimposed on the function. The set of these discrete locations gives the sampled function. However, the values of the samples still span (vertically) a continuous range of gray-level values. In order to form a digital function, the gray-level values also must be converted (quantized) into discrete quantities. The right side of Fig. 6.1 (c) shows the gray-level scale divided into eight discrete levels, ranging from black to white. The vertical tick marks indicate the specific value assigned to each of the eight gray levels. The continuous gray levels are quantized simply by assigning one of the eight discrete gray levels to each sample. The assignment is made depending on the vertical proximity of a sample to a vertical tick mark. The digital samples

resulting from both sampling and quantization are shown in Fig.6.1 (d). Starting at the top of the image and carrying out this procedure line by line produces a two-dimensional digital image.

Sampling in the manner just described assumes that we have a continuous image in both coordinate directions as well as in amplitude. In practice, the method of sampling is determined by the sensor arrangement used to generate the image. When an image is generated by a single sensing element combined with mechanical motion, as in Fig. 2.13, the output of the sensor is
quantized in the manner described above. However, sampling is accomplished by selecting the number of individual mechanical increments at which we activate the sensor to collect data.
Mechanical motion can be made very exact so, in principle; there is almost no limit as to how fine we can sample an image. However, practical limits are established by imperfections in the optics used to focus on the sensor an illumination spot that is inconsistent with the fine resolution achievable with mechanical displacements. When a sensing strip is used for image acquisition, the number of sensors in the strip establishes the sampling limitations in one image direction. Mechanical motion in the other direction can be controlled more accurately, but it makes little sense to try to achieve sampling density in one direction that exceeds the sampling limits established by the number of sensors in the other. Quantization of the sensor outputs completes the process of generating a digital image.

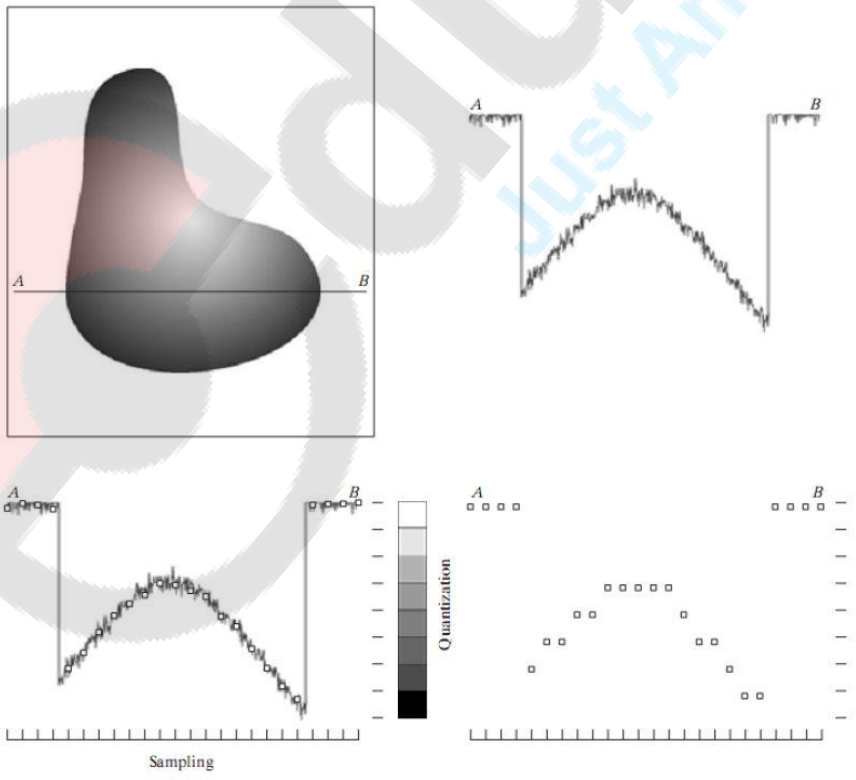
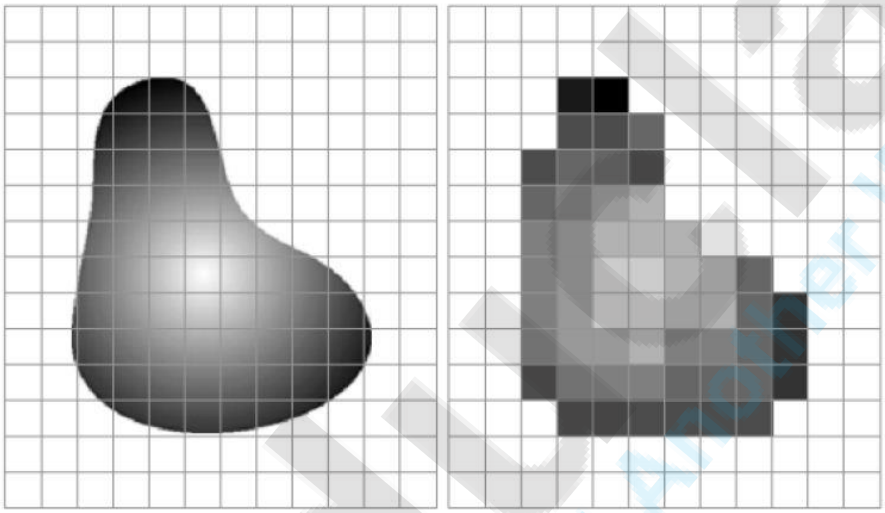


Fig.6.1. Generating a digital image (a) Continuous image (b) A scan line from A to Bin the continuous image, used to illustrate the concepts of sampling and quantization (c) Sampling and quantization. (d) Digital scan line

When a sensing array is used for image acquisition, there is no motion and the number of sensors in the array establishes the limits of sampling in both directions. Figure 6.2 illustrates this concept. Figure 6.2 (a) shows a continuous image projected onto the plane of an array sensor.
Figure 6.2 (b) shows the image after sampling and quantization. Clearly, the quality of a digital
image is determined to a large degree by the number of samples and discrete gray levels used in sampling and quantization.



a b

Fig.6.2. (a) Continuos image projected onto a sensor array (b) Result of image sampling and quantization.

5.4) Representing Digital Images

An image may be defined as a two-dimensional function, f(x, y), where x and y are spatial (plane) coordinates, and the amplitude off at any pair of coordinates (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. The field of digital image processing refers to processing digital images by means of a digital computer. Note that a digital image is composed of a finite number of elements, each of which has a particular location and value.

These elements are referred to as picture elements, image elements, peels, and pixels. Pixel is the term most widely used to denote the elements of a digital image. Vision is the most

advanced of our senses, so it is not surprising that images play the single most important role in human perception. However, unlike humans, who are limited to the visual band of the electromagnetic (EM) spectrum, imaging machines cover almost the entire EM spectrum, ranging from gamma to radio waves. They can operate on images generated by sources that humans are not accustomed to associating with images. These include ultra-sound, electron microscopy, and computer-generated images. Thus, digital image processing encompasses a wide and varied field of applications.

There is no general agreement among authors regarding where image processing stops and other related areas, such as image analysis and computer vision, start. Sometimes a distinction is made by defining image processing as a discipline in which both the input and output of a process are images. We believe this to be a limiting and somewhat artificial boundary.
For example, under this definition, even the trivial task of computing the average intensity of an image (which yields a single number) would not be considered an image processing operation. On the other hand, there are fields such as computer vision whose ultimate goal is to use computers to emulate human vision, including learning and being able to make inferences and take actions based on visual inputs. This area itself is a branch of artificial intelligence (AI) whose objective is to emulate human intelligence. The field of AI is in its earliest stages of infancy in terms of development, with progress having been much slower than originally anticipated. The area of image analysis (also called image understanding) is in between image processing and computer vision.

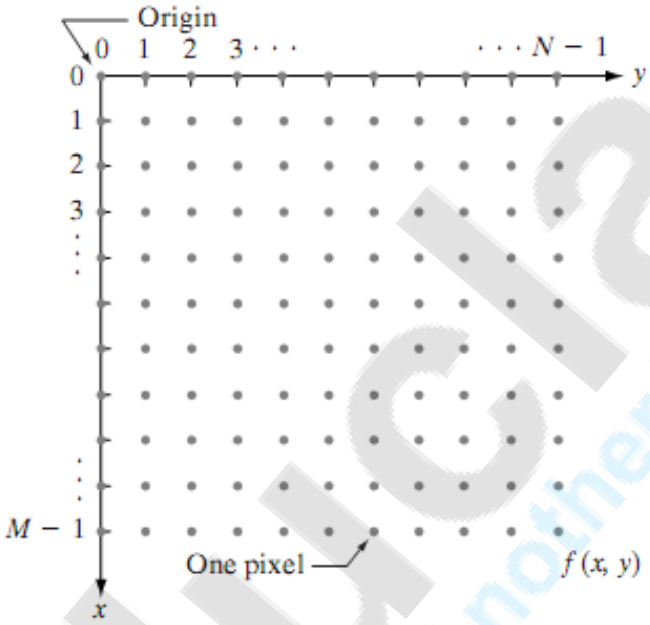
There are no clear-cut boundaries in the continuum from image processing at one end to computer vision at the other. However, one useful paradigm is to consider three types of computerized processes in this continuum: low-, mid-, and high-level processes. Lowlevel processes involve primitive operations such as image preprocessing to reduce noise, contrast enhancement, and image sharpening. A low-level process is characterized by the fact that both its inputs and outputs are images. Mid-level processing on images involves tasks such as segmentation (partitioning an image into regions or objects), description of those objects to reduce them to a form suitable for computer processing, and classification (recognition) of individual objects. A mid-level process is characterized by the fact that its inputs generally are images, but its outputs are attributes extracted from those images (e.g., edges, contours, and the identity of individual objects).

Finally, higher-level processing involves “making sense” of an ensemble of recognized objects, as in image analysis, and, at the far end of the continuum, performing the cognitive functions normally associated with vision and, in addition, encompasses processes that extract attributes from images, up to and including the recognition of individual objects. As a simple illustration to clarify these concepts, consider the area of automated analysis of text. The processes of acquiring an image of the area containing the text, preprocessing that image, extracting (segmenting) the individual characters, describing the characters in a form suitable for computer processing, and recognizing those individual characters are in the scope of what we call digital image processing.

5.5) Representing Digital Images:

We will use two principal ways to represent digital images. Assume that an image f(x, y) is sampled so that the resulting digital image has M rows and N columns. The values of the coordinates (x, y) now become discrete quantities. For notational clarity and convenience, we shall use integer values for these discrete coordinates.

Thus, the values of the coordinates at the origin are (x, y) = (0, 0). The next coordinate values along the first row of the image are represented as (x, y) = (0, 1). It is important to keep in mind that the notation (0, 1) is used to signify the second sample along the first row. It does not mean that these are the actual values of physical coordinates when the image was sampled. Figure 1 shows the coordinate convention used.



The notation introduced in the preceding paragraph allows us to write the complete M*N digital image in the following compact matrix form:

$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0, N-1) \\ f(1,0) & f(1,1) & \cdots & f(1, N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1, N-1) \end{bmatrix}.$$

The right side of this equation is by definition a digital image. Each element of this matrix array is called an image element, picture element, pixel, or pel.

5.6) Spatial and Gray – Level Resolution

Image resolution

Image resolution can be defined in many ways.One type of it which is pixel resolution that has been discussed in the tutorial of pixel resolution and aspect ratio.

In this tutorial, we are going to define another type of resolution which spatial resolution is.

5.6.1) Spatial resolution:

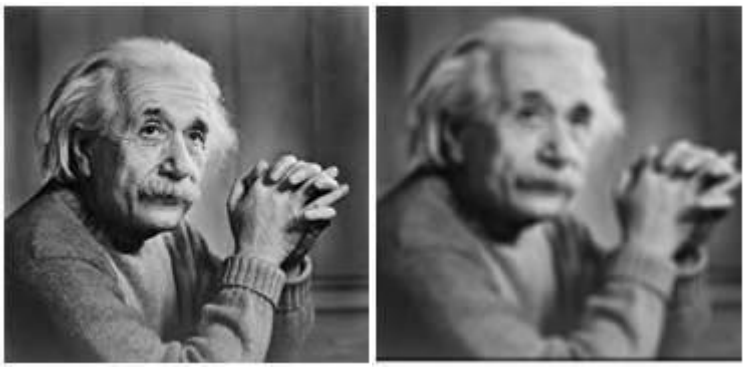
Spatial resolution states that the clarity of an image cannot be determined by the pixel resolution. The number of pixels in an image does not matter.

Spatial resolution can be defined as the smallest discernible detail in an image. (Digital Image Processing - Gonzalez, Woods - 2nd Edition) Or in other way we can define spatial resolution as the number of independent pixels values per inch. In short what spatial resolution refers to is that we cannot compare two different types of images to see that which one is clear or which one is not.If we have to compare the two images, to see which one is more clear or which has more spatial resolution, we have to compare two images of the same size.

For example:

You cannot compare these two images to see the clarity of the image.

Although both images are of the same person , but that is not the condition we are judging on. The picture on the left is zoomed out picture of Einstein with dimensions of 227 x 222. Whereas the picture on the right side has the dimensions of 980 X 749 and also it is a zoomed image. We cannot compare them to see that which one is more clear. Remember the factor of zoom does not matter in this condition, the only thing that matters is that these two pictures are not equal. So in order to measure spatial resolution, the pictures below would server the purpose.



Now you can compare these two pictures. Both the pictures has same dimensions which are of 227 X 222. Now when you compare them , you will see that the picture on the left side has more spatial resolution or it is more clear then the picture on the right side. That is because the picture on the right is a blurred image.

MEASURING SPATIAL RESOLUTION:

Since the spatial resolution refers to clarity , so for different devices , different measure has been made to measure it.

FOR EXAMPLE:

1. Dots per inch
2. Lines per inch
3. Pixels per inch

They are discussed in more detail in the next tutorial but just a brief introduction has been given below.

DOTS PER INCH:

Dots per inch or DPI is usually used in monitors.

LINES PER INCH:

Lines per inch or LPI is usually used in laser printers.

PIXEL PER INCH:

Pixel per inch or PPI is measure for different devices such as tablets , Mobile phones e.t.c.

5.6.2) Gray Level Resolution

Image resolution:

A resolution can be defined as the total number of pixels in an image. This has been discussed in Image resolution. And we have also discussed, that clarity of an image doesnot depends on number of pixels, but on the spatial resolution of the image. This has been discussed in the spatial resolution. Here we are going to discuss another type of resolution which is called gray level resolution.

Gray level resolution:

Gray level resolution refers to the predictable or deterministic change in the shades or levels of gray in an image. In short gray level resolution is equal to the number of bits per pixel.

We have already discussed bits per pixel in our tutorial of bits per pixel and image storage requirements. We will define bpp here briefly.

BPP:

The number of different colors in an image is depends on the depth of color or bits per pixel.

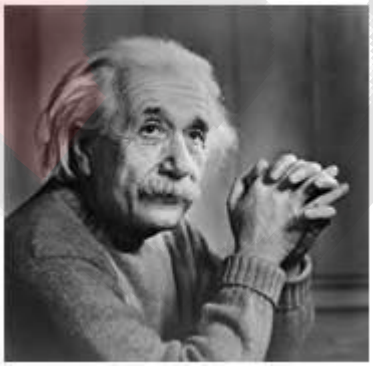
MATHEMATICALLY:

The mathematical relation that can be established between gray level resolution and bits per pixel can be given as.

$$L = 2^k$$

In this equation L refers to number of gray levels. It can also be defined as the shades of gray. And k refers to bpp or bits per pixel. So the 2 raise to the power of bits per pixel is equal to the gray level resolution.

For example:



The above image of Einstein is an gray scale image. Means it is an image with 8 bits per pixel or 8bpp.

Now if were to calculate the gray level resolution, here how we gonna do it.

$$L = 2^k$$

$$\text{Where } k = 8$$

$$L = 2^8$$

$$L = 256.$$

It means it gray level resolution is 256. Or in other way we can say that this image has 256 different shades of gray. The more is the bits per pixel of an image , the more is its gray level resolution.

Defining gray level resolution in terms of bpp:

It is not necessary that a gray level resolution should only be defined in terms of levels. We can also define it in terms of bits per pixel.

FOR EXAMPLE:

If you are given an image of 4 bpp , and you are asked to calculate its gray level resolution. There are two answers to that question. The first answer is 16 levels. The second answer is 4 bits.

FINDING BPP FROM GRAY LEVEL RESOLUTION:

You can also find the bits per pixels from the given gray level resolution. For this , we just have to twist the formula a little.

Equation 1.

$$L = 2^k$$

This formula finds the levels. Now if we were to find the bits per pixel or in this case k, we will simply change it like this.

K = log base 2(L) Equation (2)

Because in the first equation the relationship between Levels (L) and bits per pixel (k) is exponentional. Now we have to revert it , and thus the inverse of exponentional is log. Lets take an example to find bits per pixel from gray level resolution.

FOR EXAMPLE:

If you are given an image of 256 levels. What is the bits per pixel required for it.

Putting 256 in the equation , we get.

K = log base 2 (256)

K = 8.

So the answer is 8 bits per pixel.

Gray level resolution and quantization:

The quantization will be formally introduced in the next tutorial, but here we are just going to explain the relation ship between gray level resolution and quantization.

Gray level resolution is found on the y axis of the signal. In the tutorial of Introduction to signals and system, we have studied that digitizing an analog signal requires two steps.Sampling and quantization.

Sampling is done on x axis. And quantization is done in Y axis. So that means digitizing the gray level resolution of an image is done in quantization.

UNIT 6

IMAGE ENHANCEMENT TECHNIQUES

6.1)Image Enhancement in the Spatial Domain

Suppose we have a digital image which can be represented by a two dimensional random field f(x, y).

An image processing operator in the spatial domain may be expressed as a mathematical function T[.] applied to the image f(x, y) to produce a new image g(x,y) = T[f(x, y)] as follows.

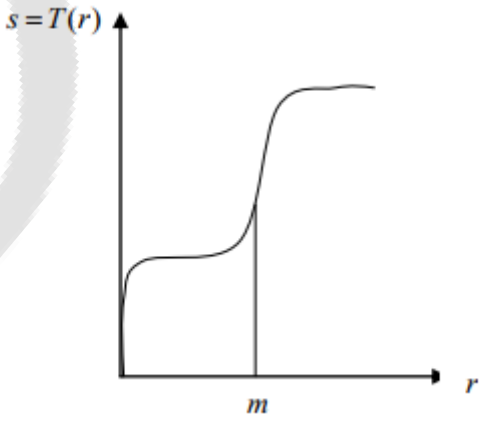
$$g(x, y) = T[f(x, y)]$$

The operator T applied on f(x, y) may be defined over:

I. A single pixel (x, y). In this case T is a grey level transformation (or mapping) function.
II. Some neighbourhood of (x, y).
III. T may operate to a set of input images instead of a single image.

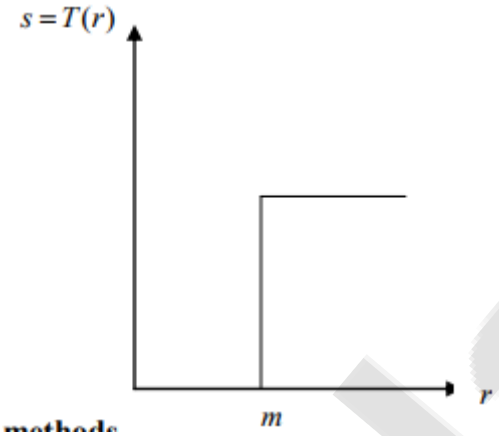
Example 1

The result of the transformation shown in the figure below is to produce an image of higher contrast than the original, by darkening the levels below m and brightening the levels above m in the original image. This technique is known as **contrast stretching.**



Example 2

The result of the transformation shown in the figure below is to produce a binary image.



Frequency domain methods

Let g(x, y)be a desired image formed by the convolution of an image f(x, y)and a linear, position invariant operator h(x, y), that is:

$$G(x, y) = h(x, y) * f(x, y)$$

The following frequency relationship holds:

$$G(u, v) = H(u, v)F(u, v)$$

We can select H(u, v) so that the desired image

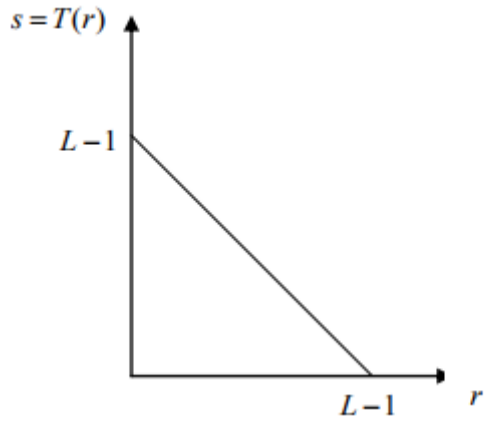
$$g(x, y) = \Im^{-1}\{H(u, v)\ F(u, v)\}$$

exhibits some highlighted features of f(x, y) . For instance, edges in f(x, y) can be accentuated by using a function H(u, v) that emphasises the high frequency components ofF(u, v).

6.1.1) Image Negatives

The negative of a digital image is obtained by the transformation function s = T(r)= L –1– r shown in the following figure, where L is the number of grey levels. The idea is that the intensity of the output image decreases as the

intensity of the input increases. This is useful in numerous applications such as displaying medical images.



The negative of in image with gray level in the range [0, l-1] is obtained by using the negative transformation. The expression of the transformation is

S= L-1-r

Reverting the intensity levels of an image in this manner produces the equivalent of a photographic negative. This type of processing is practically suited for enhancing white or gray details embedded in dark regions of an image especially when the black areas are dominant in size.

6.1.2) Log Transformations

Log Transformation is defined as S = C log (1+r) where C is any constant. This transformation enhances the small magnitude pixels compared to those pixels with large magnitudes.

This transformation is useful when the dynamic range of some processed pixels exceeds the capability of the display device. For example, The dynamic range of unilaterlity transformed (i.e Fourier Transform) is so large that only the brighest points are visible on the display screen.

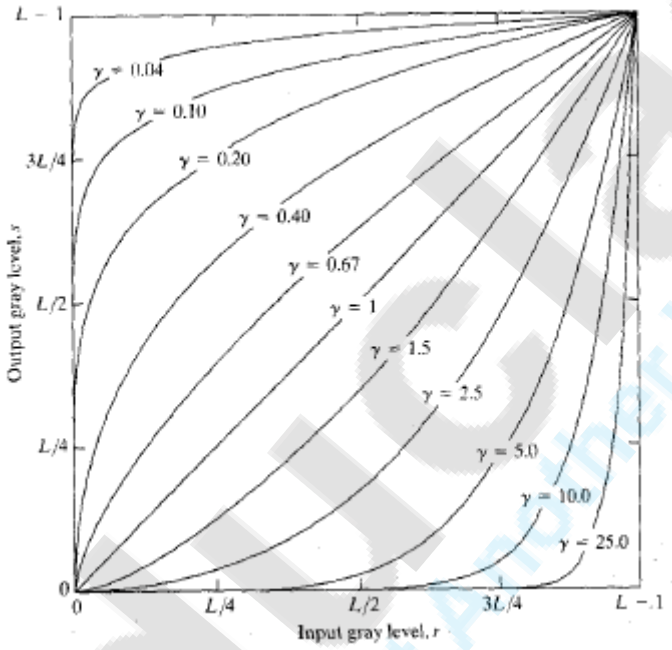
A effective way to overcome this problem is to compress the dynamic range of pixel values via the log transformations.

6.1.3) Power Law Transformation

Power law transformations has the basic form

$$S=cr^{\gamma}$$

Where c and y are positive constants. Power law curves with fractional values of y map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher values of input gray levels. We may get various curves by varying values of y.



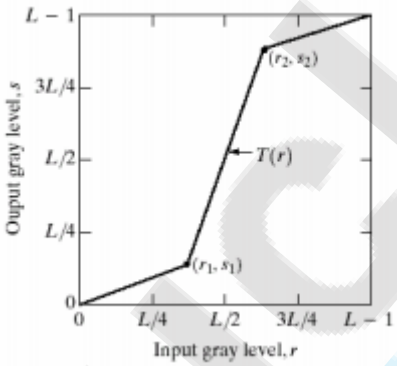
A variety of devices used for image capture, printing and display respond according to a power law. The process used to correct this power law response phenomenon is called gamma correction. For eg-CRT devices have intensity to voltage response that is a power function. Gamma correction is important if displaying an image accurately on a computer screen is of concern. Images that are not corrected properly can look either bleached out or too dark. Color phenomenon also uses this concept of gamma correction. It is becoming more popular due to use of images over the internet. It is important in general purpose contract manipulation. To make an image black we use y>1 and y

6.2) Piecewise-Linear Transformation Functions

The principal advantage of piecewise linear functions is that these functions can be arbitrarily complex. But their specification requires considerably more user input.

6.2.1)Contrast Stretching

It is the simplest piecewise linear transformation function. We may have various low contrast images and that might result due to various reasons such as lack of illumination, problem in imaging sensor or wrong setting of lens aperture during image acquisition. The idea behind contrast stretching is to increase the dynamic range of gray levels in the image being processed.



Applied contrast stretching: (r1 ,s1) = (rmin,0) and (r2 ,s2) = (rmax,L-1)

The location of points (r1,s1) and (r2,s2) control the shape of the curve

a) If r1=r2 and s1=s2, the transformation is a linear function that deduces no change in gray levels.
b) If r1=s1, s1=0 , and s2=L-1, then the transformation become a thresholding function that creates a binary image
c) Intermediate values of (r1, s1) and (r2, s2) produce various degrees of spread in the gray value of the output image thus effecting its contract.

Generally r1≤ r2 and s1 ≤ s2 so that the function is single valued and monotonically increasing

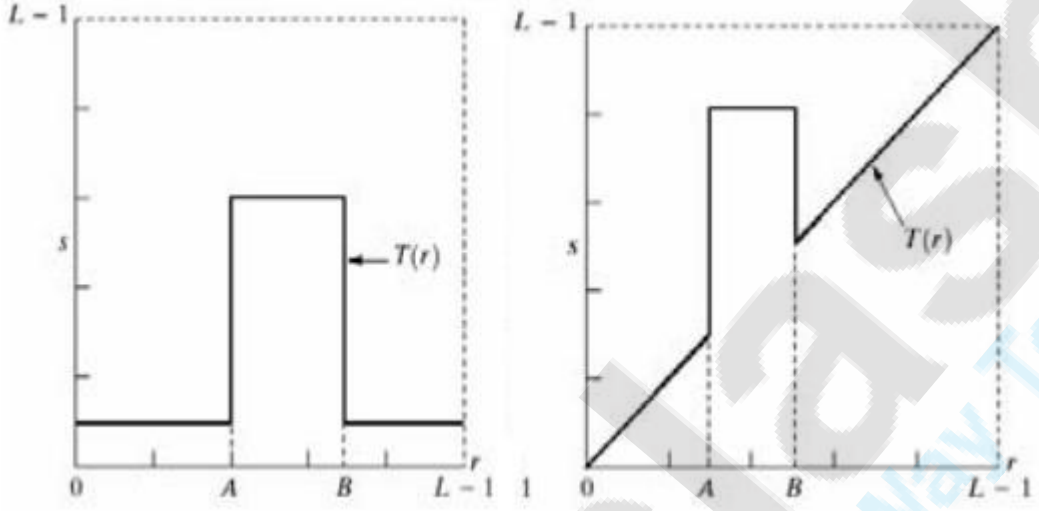
6.2.2) Gray-level slicing

Highlighting a specific range of gray levels in an image is often desirable

For example when enhancing features such as masses of water in satellite image and enhancing flaws in x- ray images.

There are two ways of doing this

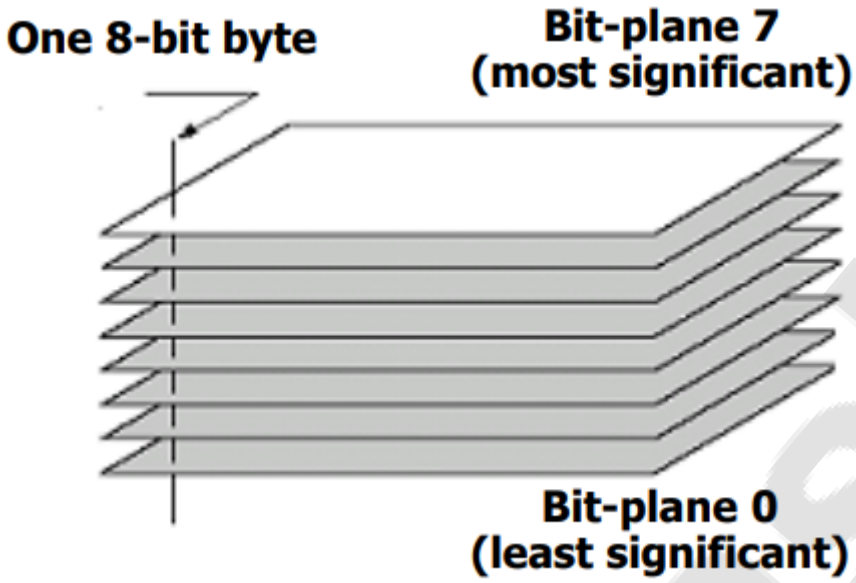
a) One method is to display a high value for all gray level in the range.Of interest and a low value for all other gray level.



b) Second method is to brighten the desired ranges of gray levels but preserve the background and gray level tonalities in the image.

6.2.3) Bit plane slicing

Sometimes it is important to highlight the contribution made to the total image appearance by specific bits. Suppose that each pixel is represented by 8 bits. Imagine that an image is composed of eight 1-bit planes ranging from bit plane 0 for the least significant bit to bit plane 7 for the most significant bit. In terms of 8-bit bytes, plane 0 contains all the lowest order bits in the image and plane 7 contains all the high order bits.



High order bits contain the majority of visually significant data and contribute to more subtle details in the image. Separating a digital image into its bits planes is useful for analyzing the relative importance played by each bit of the image. It helps in determining the adequacy of the number of bits used to quantize each pixel. It is also useful for image compression.

Highlighting a specific bit of every pixel in an image is called Bit Level Slicing. Consider an image with 8 bit pixels values composed of eight 1 bit planes ranging from bit plane 0.For the LSB to bit plane 7 for the MSB. In terms of 8 bit bytes, plane 0 contains all the lowest order bits of every pixel and plane-7 contains all the high order is MSB bits of every pixel.
The high order bits (Top four) contain the majority of the visually significant data. The other bit planes contribute to more subtle details in the image.

Application: Separating a digital image into its bit planes is useful for analysing the relative importance played by each bit of the image. This type of decomposition is useful for image compression.

6.3) Introduction to Histogram

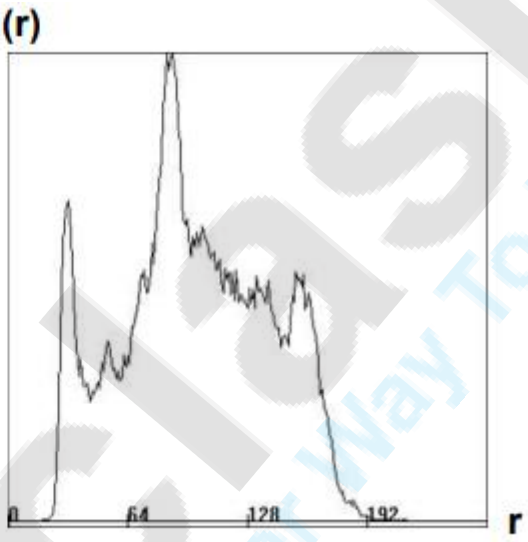
The histogram of a digital image with gray levels in the range [0, L-1] is a discrete function of the form

$$H(r_k)=n_k$$

Where rk is the kth gray level and nk is the number of pixels in the image having the level rk. A normalized histogram is given by the equation

$$p(rk)=nk/n \text{ for } k=0,1,2,.....,L-1$$

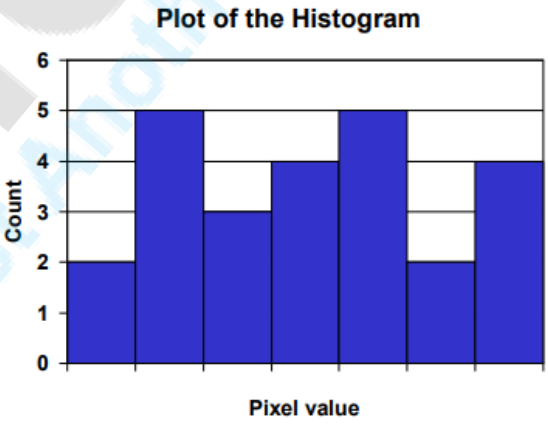
P(rk) gives the estimate of the probability of occurrence of gray level rk. The sum of all components of a normalized histogram is equal to 1. The histogram plots are simple plots of H(rk)=nk versus rk.



5x5 image

2	3	4	4	6
1	2	4	5	6
1	1	5	6	6
0	1	3	3	4
0	1	2	3	4

Graylevel	Count
0	2
1	5
2	3
3	4
4	5
5	2
6	4
Total	25



In the dark image the components of the histogram are concentrated on the low (dark) side of the gray scale. In case of bright image the histogram components are baised towards the high side of the gray scale. The histogram of a low contrast image will be narrow and will be centered towards the middle of the gray scale. The components of the histogram in the high contrast image cover a broad range of the gray scale. The net effect of this will be an image that shows a great deal of gray levels details and has high dynamic range.

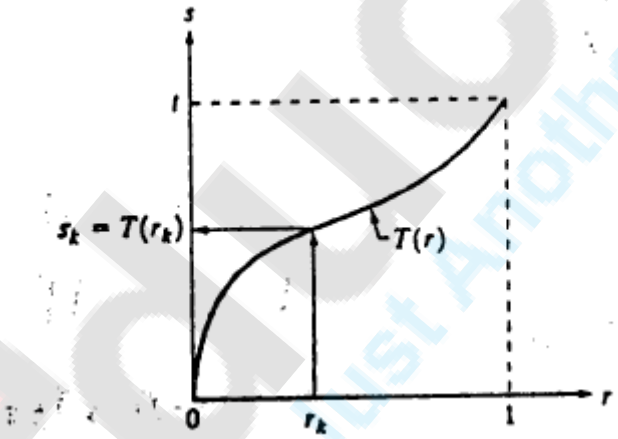
The histogram of an image represents the relative frequency of occurance of the various gray levels in the image.
Histogram modeling techniques modify an image so that its histogram has a desired shape. This is useful in stretching the low contrast levels of image with narrow histograms. The type and degree of enhancement obtained depends on the nature of the specified histogram.
Let the variable r represent the gray level of the pixels in the image to be enhanced.
Normalizing r in the range [0,1] with r=0, representing black and r=1 representing white in the gray scale. Transformation function T is such that

S = T(r)

It is assumed that the transformation function satisfies conditions:-

I. The transformation function T(r) is single valued and monotinically increasing in the interval $0 \le r \le 1$.
II. $0 \le T(r) \le 1$ for $0 \le r \le 1$.

The inverse transformation from s back to r is, r = $T^{-1}(S)$. $T^{-1}(S)$ should also satisfy the above conditions.



Before discussing the use of Histograms in image processing, we will first look at what histogram is, how it is used and then an example of histograms to have more understanding of histogram.

6.4)Histograms

A histogram is a graph. A graph that shows frequency of anything. Usually histogram have bars that represent frequency of occurring of data in the whole data set.

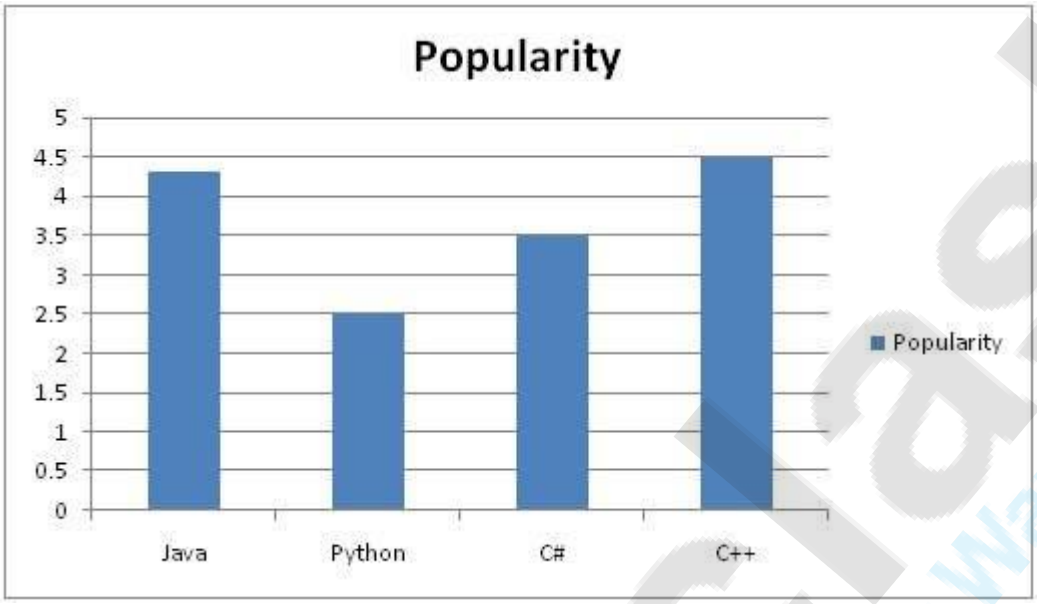
A Histogram has two axis the x axis and the y axis.

The x axis contains event whose frequency you have to count.

The y axis contains frequency.

The different heights of bar shows different frequency of occurrence of data.

Usually a histogram looks like this.



Now we will see an example of this histogram is build

Example

Consider a class of programming students and you are teaching python to them.

At the end of the semester, you got this result that is shown in table. But it is very messy and does not show your overall result of class. So you have to make a histogram of your result, showing the overall frequency of occurrence of grades in your class. Here how you are going to do it.

6.4.1)Result sheet

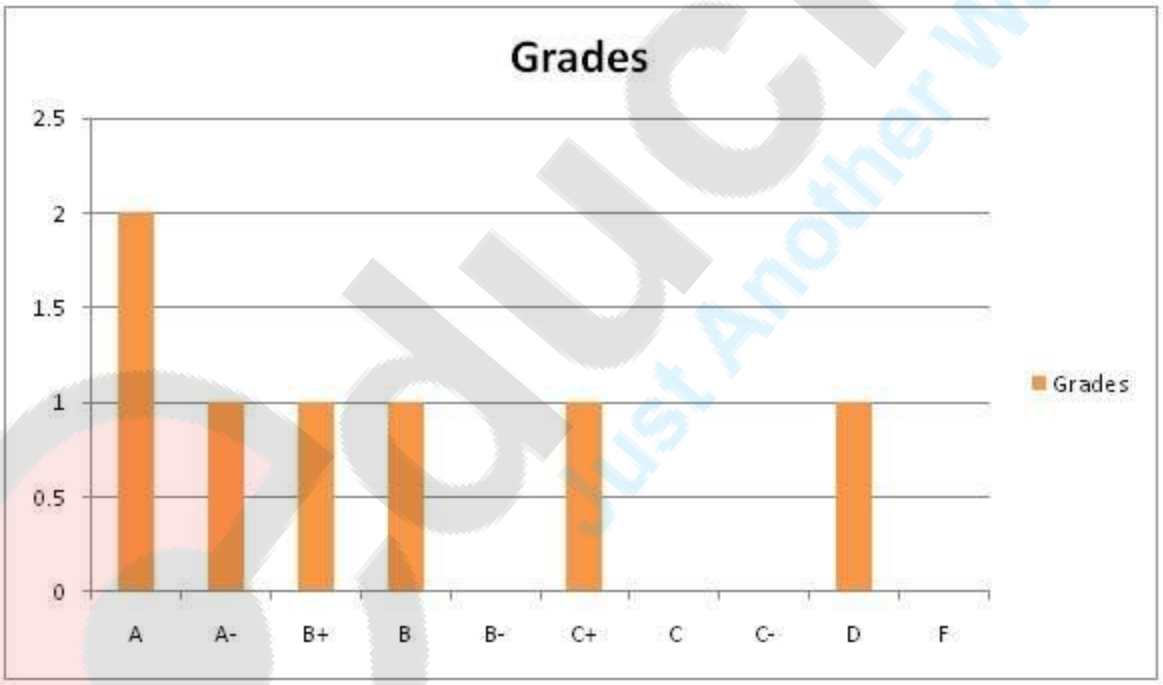
Name	Grade
John	A
Jack	D
Carter	B

Tommy	A
Lisa	C+
Derek	A-
Tom	B+

6.4.2)Histogram of result sheet

Now what you are going to do is, that you have to find what comes on the x and the y axis.

There is one thing to be sure, that y axis contains the frequency, so what comes on the x axis. X axis contains the event whose frequency has to be calculated. In this case x axis contains grades.

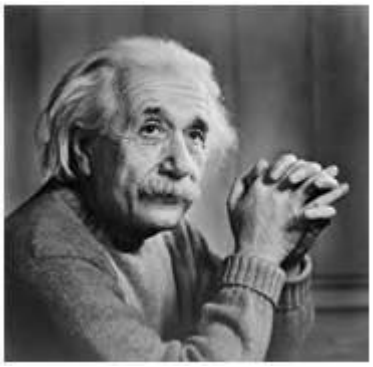


Now we will how do we use a histogram in an image.

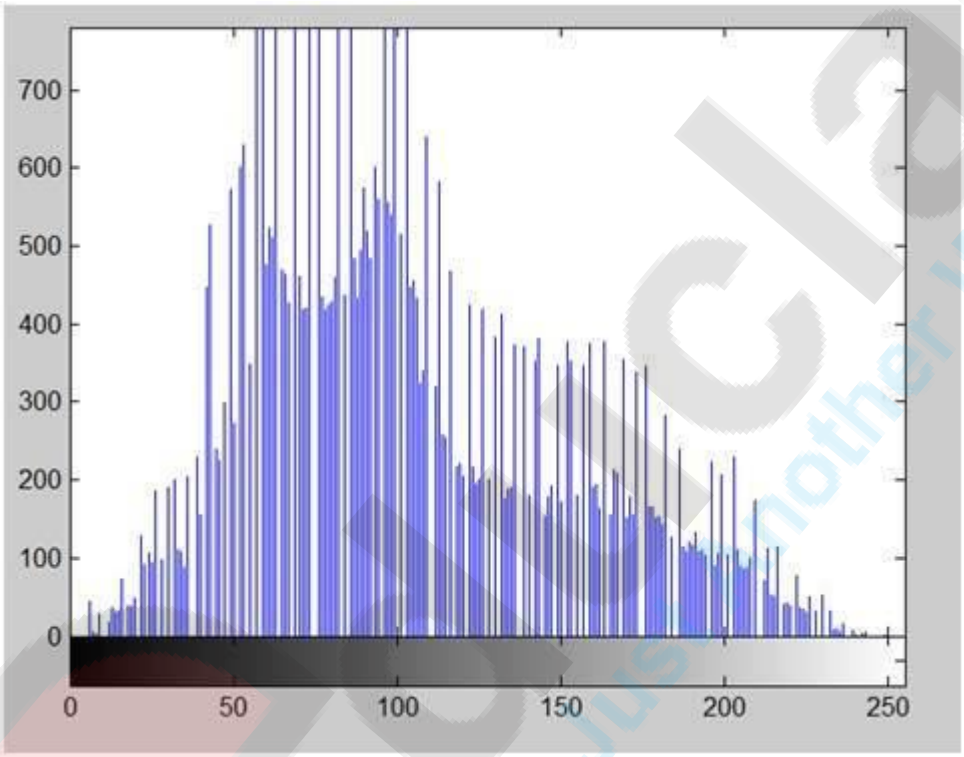
6.4.3)Histogram of an image

Histogram of an image, like other histograms also shows frequency. But an image histogram, shows frequency of pixels intensity values. In an image histogram, the x axis shows the gray level intensities and the y axis shows the frequency of these intensities.

For example



The histogram of the above picture of the Einstein would be something like this



The x axis of the histogram shows the range of pixel values. Since its an 8 bpp image, that means it has 256 levels of gray or shades of gray in it. Thats why the range of x axis starts from 0 and end at 255 with a gap of 50. Whereas on the y axis, is the count of these intensities.

As you can see from the graph, that most of the bars that have high frequency lies in the first half portion which is the darker portion. That means that the image we have got is darker. And this can be proved from the image too.

6.4.4)Applications of Histograms

Histograms has many uses in image processing. The first use as it has also been discussed above is the analysis of the image. We can predict about an image by just looking at its histogram. Its like looking an x ray of a bone of a body.

The second use of histogram is for brightness purposes. The histograms has wide application in image brightness. Not only in brightness, but histograms are also used in adjusting contrast of an image.

Another important use of histogram is to equalize an image.

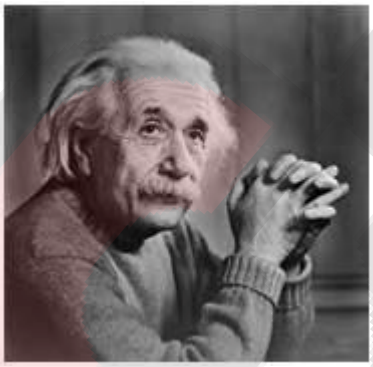
And last but not the least, histogram has wide use in thresholding. This is mostly used in computer vision.

6.5)Histogram Equalization

Histogram equalization is used to enhance contrast. It is not necessary that contrast will always be increase in this. There may be some cases were histogram equalization can be worse. In that cases the contrast is decreased.

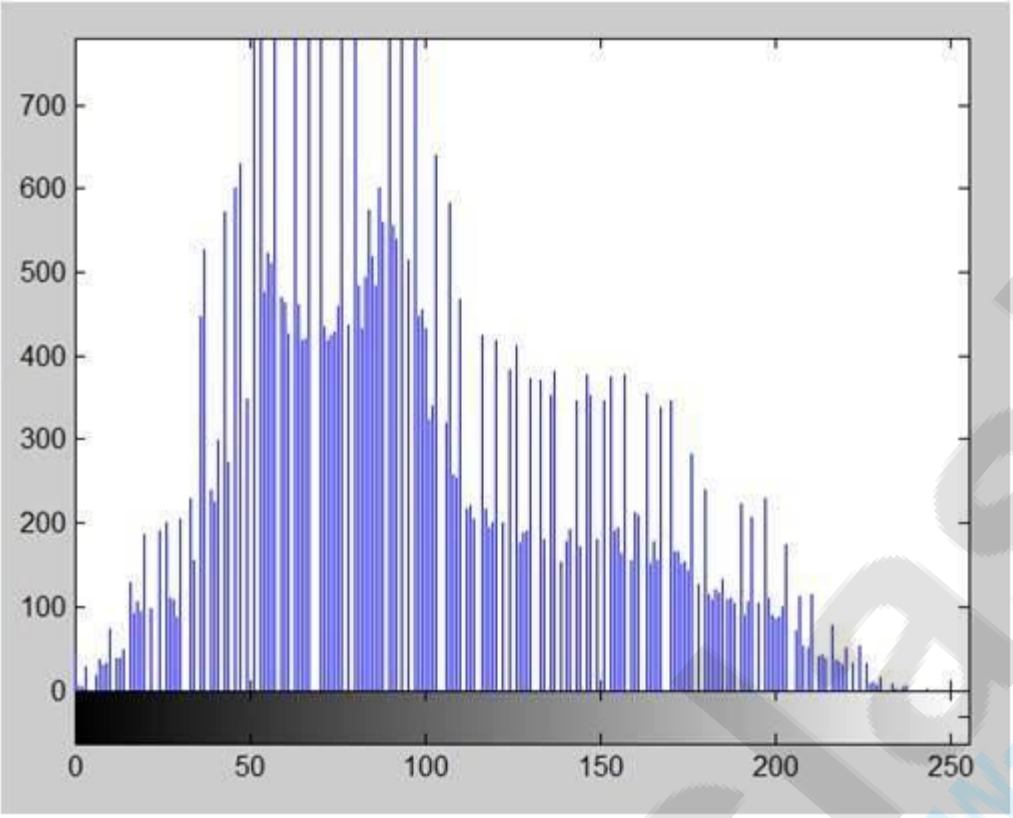
Lets start histogram equalization by taking this image below as a simple image.

Image



Histogram of this image

The histogram of this image has been shown below.



Now we will perform histogram equalization to it.

PMF

First we have to calculate the PMF (probability mass function) of all the pixels in this image. If you donot know how to calculate PMF, please visit our tutorial of PMF calculation.

CDF

Our next step involves calculation of CDF (cumulative distributive function). Again if you donot know how to calculate CDF , please visit our tutorial of CDF calculation.

Calculate CDF according to gray levels

Lets for instance consider this , that the CDF calculated in the second step looks like this.

Gray Level Value	CDF
0	0.11
1	0.22
2	0.55

3	0.66
4	0.77
5	0.88
6	0.99
7	1

Then in this step you will multiply the CDF value with (Gray levels (minus) 1).

Considering we have an 3 bpp image. Then number of levels we have are 8. And 1 subtracts 8 is 7. So we multiply CDF by 7. Here what we got after multiplying.

Gray Level Value	CDF	CDF * (Levels-1)
0	0.11	0
1	0.22	1
2	0.55	3
3	0.66	4
4	0.77	5
5	0.88	6
6	0.99	6
7	1	7

Now we have is the last step, in which we have to map the new gray level values into number of pixels.

Lets assume our old gray levels values has these number of pixels.

Gray Level Value	Frequency
0	2
1	4
2	6
3	8

4	10
5	12
6	14
7	16

Now if we map our new values to , then this is what we got.

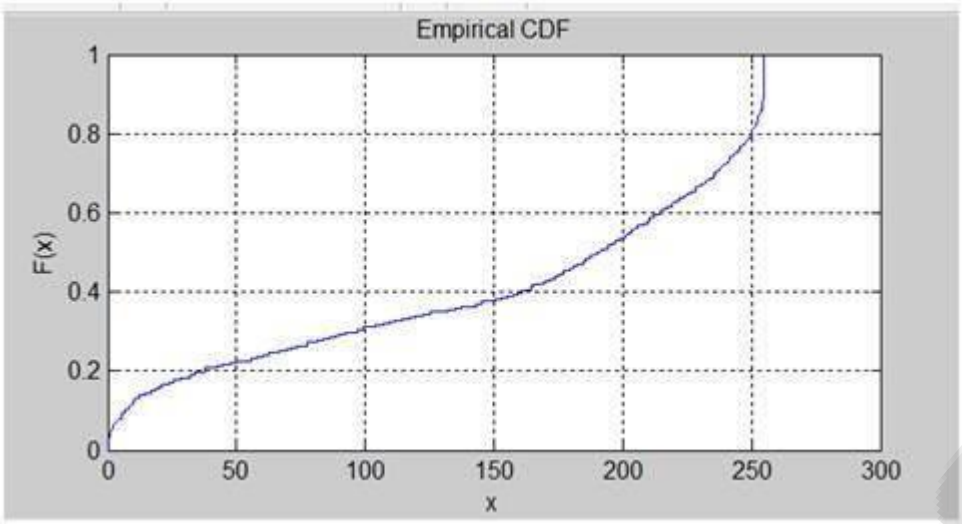
Gray Level Value	New Gray Level Value	Frequency
0	0	2
1	1	4
2	3	6
3	4	8
4	5	10
5	6	12
6	6	14
7	7	16

Now map these new values you are onto histogram, and you are done.

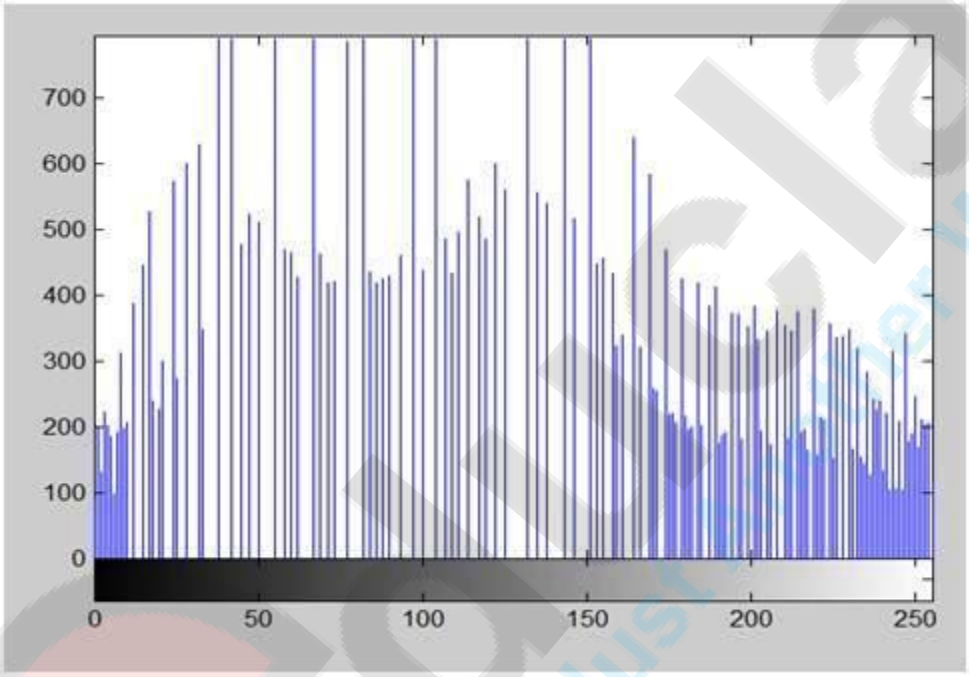
Lets apply this technique to our original image. After applying we got the following image and its following histogram.

Histogram Equalization Image

Cumulative Distributive function of this image

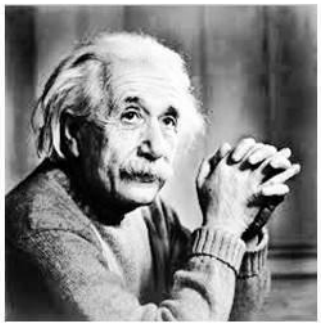


Histogram Equalization histogram

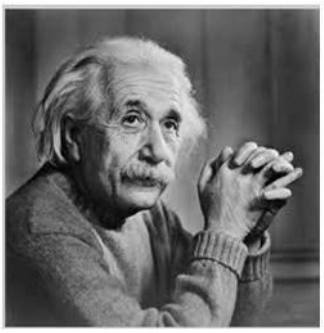


Comparing both the histograms and images

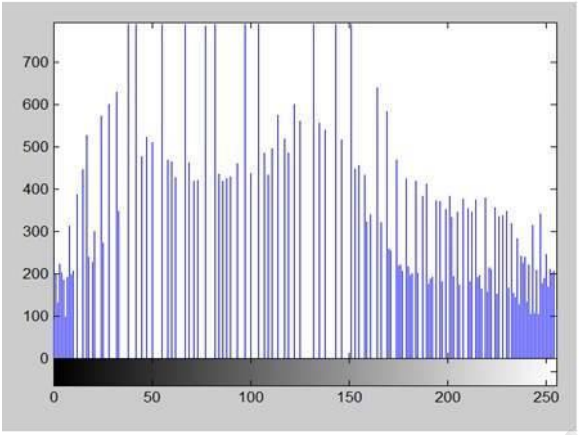
New Image



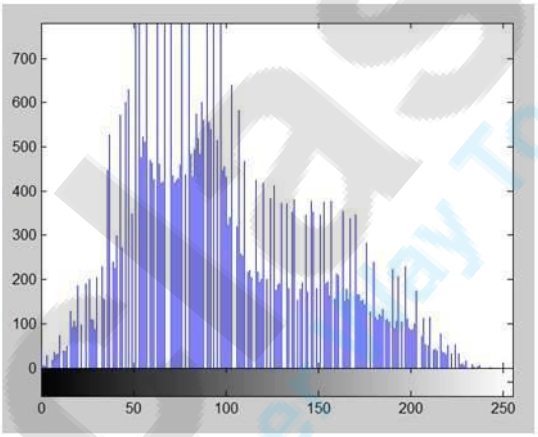
Old image



New Histogram



Old Histogram



Conclusion

As you can clearly see from the images that the new image contrast has been enhanced and its histogram has also been equalized. There is also one important thing to be note here that during histogram equalization the overall shape of the histogram changes, where as in histogram stretching the overall shape of histogram remains same.

6.6) Image Subtraction

The difference between two images f(x,y) and h(x,y) is expressed as

$$g(x,y)= f(x,y)-h(x,y)$$

It is obtained by computing the difference between all pairs of corresponding pixels from f and h. The key usefulness of subtraction is the enhancement of difference between images. This concept is used in another gray scale transformation for enhancement known as bit plane slicing The higher order bit planes of an image carry a significant amount of visually relevant detail while the lower planes contribute to fine details. It we subtract the four least significant bit planes from the image The result will be nearly identical but

there will be a slight drop in the overall contrast due to less variability in the gray level values of image

First Subtraction: $g(x,y) = f_1(x,y) - f_2(x,y)$ ← Would be in range -255 to +255

Map everything back to the range [0, 255]

$$m_0 = \min[g(x,y)]$$

$$g_1(x,y) = g(x,y) - m_0$$

$$m_1 = \max[g_1(x,y)]$$

$$g_2(x,y) = \frac{255}{m_1} g_1(x,y) = \frac{255}{m_1}\left(g(x,y) - m_0\right)$$

The use of image subtraction is seen in medical imaging area named as mask mode radiography. The mask h (x,y) is an X-ray image of a region of a patient's body this image is captured by using as intensified TV camera located opposite to the x-ray machine then a consistent medium is injected into the patient's blood storm and then a series of image are taken of the region same as h(x,y). The mask is then subtracted from the series of incoming image. This subtraction will give the area which will be the difference between f(x,y) and h(x,y) this difference will be given as enhanced detail in the output image. This procure produces a move shoving now the contrast medium propagates through various arteries of the area being viewed. Most of the image in use today is 8- bit image so the values of the image lie in the range 0 to 255. The value in the difference image can lie from -255 to 255. For these reasons we have to do some sort of scaling to display the results There are two methods to scale an image (i) Add 255 to every pixel and then divide at by 2. This gives the surety that pixel values will be in the range 0 to 255 but it is not guaranteed whether it will cover the entire 8 – bit range or not. It is a simple method and fast to implement but will not utilize the entire gray scale range to display the results. (ii) Another approach is (a) Obtain the value of minimum difference (b) Add the negative of minimum value to the pixels in the difference image(this will give a modified image whose minimum value will be 0) (c) Perform scaling on the difference image by multiplying each pixel by the quantity 255/max. This approach is complicated and difficult to implement. Image subtraction is used in segmentation application also.

Applications

1. In Compression, error image is obtained by subtracting predicted image from the original image. Error image this obtained is coded with min. number of bits.
2. Automated Inspection of PCB, to detect the missing components during soldering.
3. **Mask mode radiography-** In this area of medical application, the blood stream is injected with a radio-opaque dye and x-ray images are taken before and after the injection. The difference of the two images yields a clear display of blood flow paths.

6.7) Image Averaging

Suppose that we have an image f (x, y) of size M ×N pixels corrupted by noise n(x, y) ,so we obtain a noisy image as follows.

$$g(x, y) = f(x, y) + n(x, y)$$

For the noise process n(x, y) the following assumptions are made.

I. The noise process n(x, y) is ergodic.

II. It is zero mean, i.e., E{n(x, y)} = $\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} n(x,y) = 0$

III. It is white, i.e., the autocorrelation function of the noise process defined as R[k, l] = E{n(x, y)n(x+k,y+l)} = $\frac{1}{(M-k)(N-l)}\sum_{x=0}^{M-1-k}\sum_{y=0}^{N-1-l} n(x,y)n(x+k,y+l)$ is zero, apart for the pair [k,l] =[0,0]. Therefore, R[k, l] = $\frac{1}{(M-k)(N-l)}\sum_{x=0}^{M-1-k}\sum_{y=0}^{N-1-l} n(x,y)n(x+k,y+l) = \sigma^2_{n(x,y)}\delta(k,l)$ where $\sigma^2_{n(x,y)}$is the variance of noise.

Consider a noisy image g(x,y) formed by the addition of noise n(x,y) to the original image f(x,y)

$$g(x,y) = f(x,y) + n(x,y)$$

Assuming that at every point of coordinate (x,y) the noise is uncorrelated and has zero average value The objective of image averaging is to reduce the noise content by adding a set of noise images, {gi(x,y)} If in image formed by image averaging K different noisy images

$$\bar{g}(x,y) = \frac{1}{k}\sum_{i=1}^{K} g_i(x,y)$$

$$E\{\bar{g}(x,y)\} = f(x, y)$$

As k increases the variability (noise) of the pixel value at each location (x,y) decreases E{g(x,y)} = f(x,y) means that g(x,y) approaches f(x,y) as the number

of noisy image used in the averaging processes increases Image averaging is important in various applications such as in the field of astronomy where the images are low light levels

Eduklash Just Another Way To Learn