[**http://www.gomezconsultants.com/CSE5280/Transformations/2DTrans.html**](http://www.gomezconsultants.com/CSE5280/Transformations/2DTrans.html)

**line equation =**(y-y1) = m(x-x1)

**slope m** = (y2 - y1 ) / (x2 - x1)

**DDA Algorithm (Incremental Algorithm):-**

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) and (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));

}

**Problems:**

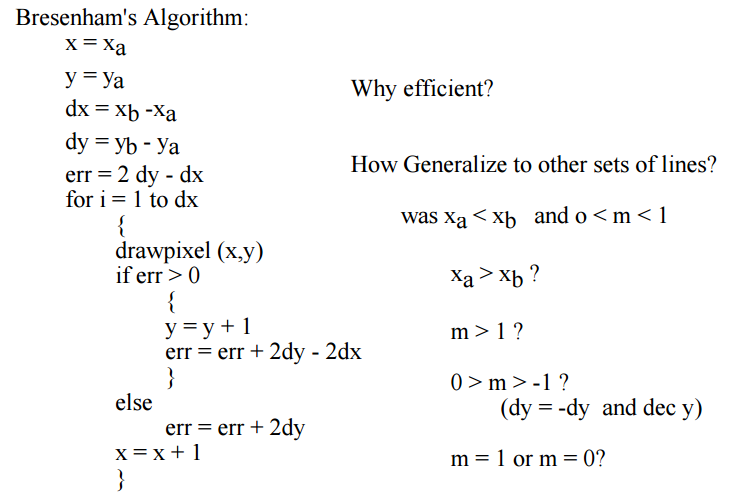
* still slow
* still floating point math

**Advantages:**

* mathematically well defined
* no spotty lines

**Bresenham’s Mid–Point Line Algorithm :-**

Given this, J. E. Bresenham developed an algorithm for line drawings that uses integer arithmetic only



It is IMPORTANT to note that this algorithm only works for lines where x1 > x0 and for slope (dy/dx) between 0 and 1.0. You must modify the algorithm to work for the other cases as well. Further, take care to handle the case of infinite slope lines (x1 == x0).

**Bresenham’s Midpoint Circle Algorithm.**

This algorithm assumes the circle center is at (0,0).

void MidpointCircle (int r)

{

int x = 0;

int y = radius;

double d = 5.0/4.0 - radius;

SetPixel(x,y);

while(y > x)

{

if (d < 0)

{

d += 2.0 \* x + 3.0; // Select East

}

else

{

d += 2.0 \* (x-y) + 5.0; // Select SE

y--;

}

x++;

SetPixel(x, y);

}

}

It is IMPORTANT to note that this algorithm only draws one–eighth of a circle, in the range of 0 to 45 degrees. You must modify the algorithm to draw properly the remaining seven eighth’s of the circle.

**Bresenham’s Midpoint Circle Algorithm.**

We cannot display a continuous arc on the raster display. Instead, we have to choose the nearest pixel position to complete the arc.

From the following illustration, you can see that we have put the pixel at (X, Y) location and now need to decide where to put the next pixel − at North (X+1, Y) or at South (X+1, Y-1).

This can be decided by the decision parameter d.

If d <= 0, then N(X+1, Y) is to be chosen as next pixel.

If d > 0, then S(X+1, Y-1) is to be chosen as the next pixel.

Now for each pixel, we will do the following operations:

1.Set initial values of (xc, yc) and (x, y)

2.Set decision parameter d to d = 3 – (2 \* r).

3.Repeat steps 4 to 8 until x < = y

4.calldrawCircle(int xc, intyc, int x, int y) function.

5.Increment value of x.

6.If d < 0, set d = d + (4\*x) + 6

7.Else, set d = d + 4 \* (x – y) + 10 and decrement y by 1.

8.calldrawCircle(int xc, intyc, int x, int y) function

**2D Translations:-**

**Translation :-**

A translation moves an object to a different position on the screen.

You can translate a point in 2D by adding translation coordinate (tx, ty) to the original coordinate (X, Y) to get the new coordinate (X’, Y’).

From the above figure, you can write that −

X’ = X + tx

Y’ = Y + ty

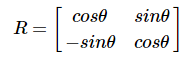
**Rotation :-**

In rotation, we rotate the object at particular angle θ (theta) from its origin.

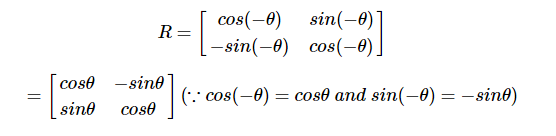
Where R is the rotation matrix

The rotation angle can be positive and negative.

For positive rotation angle, we can use the above rotation matrix.



However, for negative angle rotation, the matrix will change as shown below –



**Scaling:-**

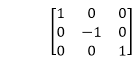
Increase or Decrease the size of the object by multiplying the original coordinates of the object with the scaling factor to get the desired result.

This can be mathematically represented as shown below −

X' = X . SX and Y' = Y . SY

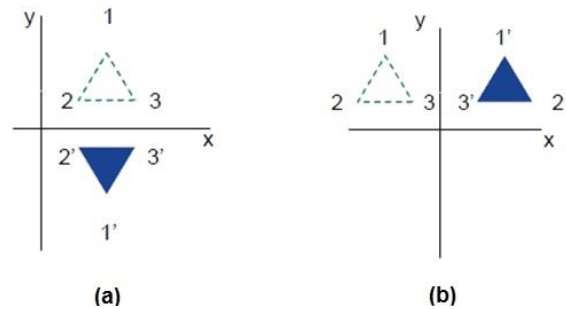
**Reflection:-**

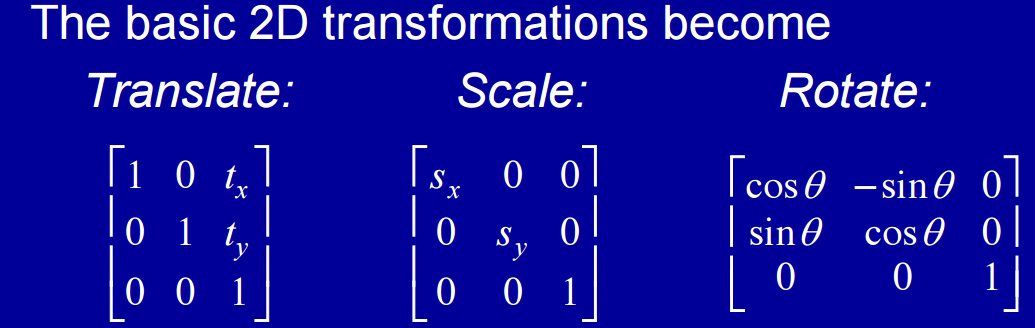
Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180°

**Reflection about x-axis:** The object can be reflected about x-axis with the help of the following matrix 

**Reflection about y-axis:** The object can be reflected about y-axis with the help of following transformation matrix Reflection

Ref https://www.javatpoint.com/computer-graphics-reflection





**2D Rotation with pivot points :-**

The position (xr, yr) of the pivot point about which the object is to be rotated.

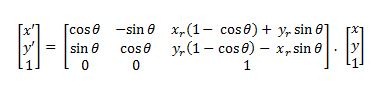
The Original position is x, y.

1.Translate the object so that the pivot-point position is moved to the coordinate origin.

2.Rotate the object about the coordinate origin.

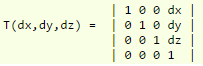
3.Translate the object so that the pivot point is returned to its original position.

The formula is below.

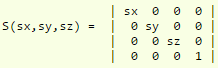
****

**3D Translations:-**

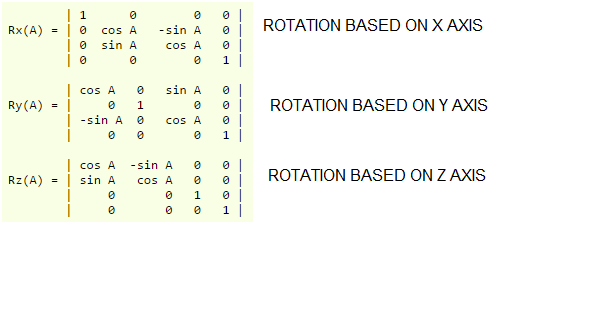
**Translation :-**

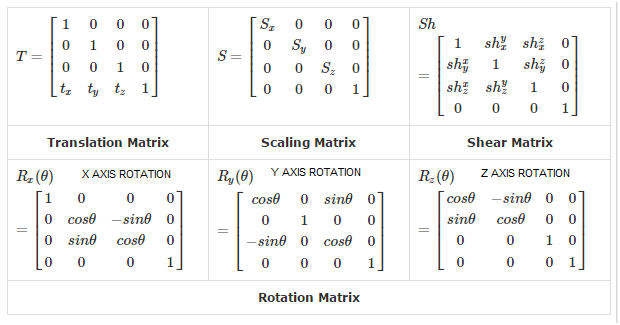
****

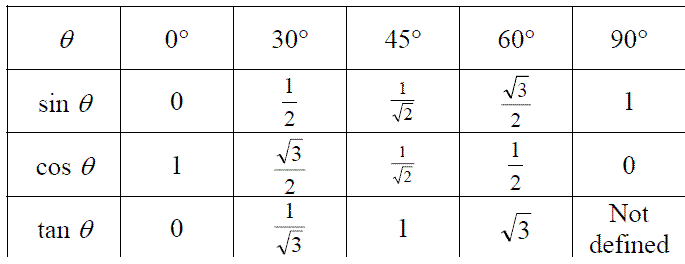
**Scaling:-**

****

**Rotation :-**



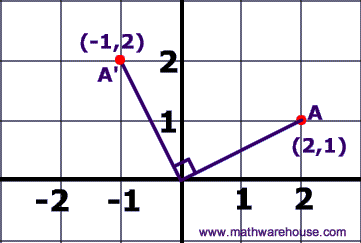




**Examples of the most common rotations**

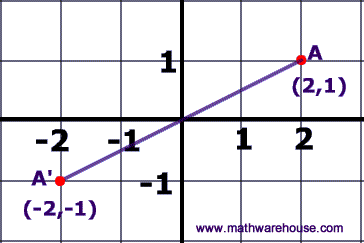
**Rotation by 90° about the origin: R(origin, 90°)**

A rotation by 90° about the origin can be seen in the picture below in which A is rotated to its image A'. The general rule for a rotation by 90° about the origin is (A,B) (-B, A)



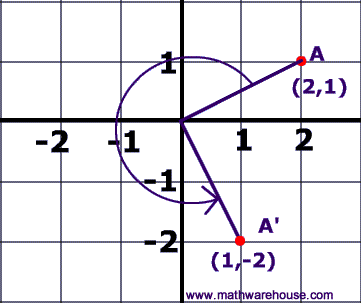
**Rotation by 180° about the origin: R(origin, 180°)**

A rotation by 180° about the origin can be seen in the picture below in which A is rotated to its image A'. The general rule for a rotation by 180° about the origin is (A,B) (-A, -B)



**Rotation by 270° about the origin: R(origin, 270°)**

A rotation by 270° about the origin can be seen in the picture below in which A is rotated to its image A'. The general rule for a rotation by 270° about the origin is (A,B) (B, -A)



**Pivot-Point Rotation:** With a graphics package that only provides a rotate function for revolving objects about the coordinate origin, we can generate rotations about any selected pivot point (x, y,) by performing the following sequence of translate-rotate-translate-operations:

1. Translate the object so that the pivot-point position is moved to the coordinate origin.

2. Rotate the object about the coordinate origin.

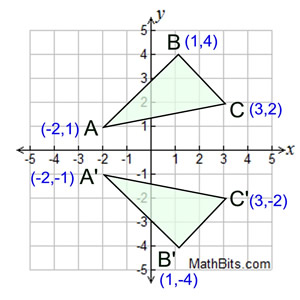
3. Translate the object so that the pivot point is returned to its original position.

Ref https://www.csee.umbc.edu/~rheingan/435/pages/res/gen-6.2Dtform-page-7m.html

**Reflect over the x-axis:**

When you reflect a point across the x-axis, the x-coordinate remains the same, but the y-coordinate is transformed into its opposite (its sign is changed).

**The reflection of the point (x,y) across the x-axis is the point (x,-y).**



|  |  |
| --- | --- |
| **Reflect over the y-axis:**  refX2 |  |

When you reflect a point across the y-axis, the y-coordinate remains the same, but the x-coordinate is transformed into its opposite (its sign is changed).

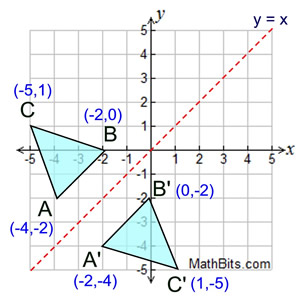
The reflection of the point (x,y) across the y-axis is the point (-x,y).

**Reflect over the y = x:**

When you reflect a point across the line *y = x,* the *x-*coordinate and *y*-coordinate change places. If you reflect over the line *y = -x*, the *x*-coordinate and *y*-coordinate change places and are negated (the signs are changed).

|  |
| --- |
| **The reflection of the point (*x,y*) across the line *y* = *x* is the point (*y, x*).** |

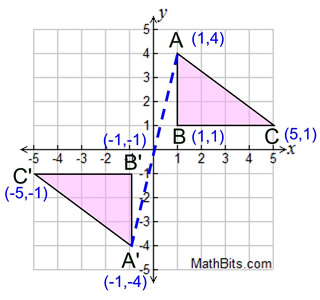
|  |
| --- |
| **The reflection of the point (*x,y*) across the line *y* = *-x* is the point (*-y, -x*).** |



**Reflection in a Point:-**

**Reflect in origin (0,0):-**

In a point reflection in the origin, the image of the point (x,y) is the point (-x,-y)



The anti-aliasing technique which allows shift of 1/4, 1/2 and 3/4 of a pixel diameter

enabling a closer path of a line is

1. Pixel phasing(yes) 2) Filtering 3) Intensity compensation 4) Sampling technique

**EXPLANATION:-**

Pixel phasing is an anti-aliasing technique, stair steps are smoothed out by moving the electron beam to more nearly approximate positions specified by the object geometry.

***Which of the following is fundamental methods of anti-aliasing?***

(a) Increase of sample rate.

(b) Treating a pixel as a finite area rather than as a point.

(c) Decrease of sample rate.

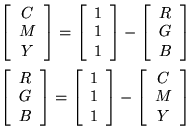
1) (a) and (b)(ANS) 2) (b) and (c) 3) (a) only 4) (b) only

**Homogeneous coordinates**

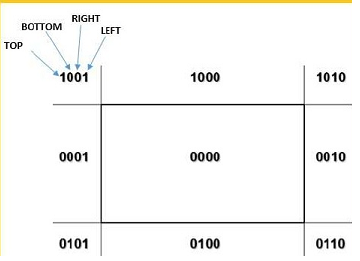
Suppose we have a point (x,y) in the Euclidean plane. To represent this same point in the projective plane, we simply add a third coordinate of 1 at the end: (x, y, 1).

(Thus the point (0,0,0) is disallowed). Because scaling is unimportant, the coordinates (X,Y,W) are called the homogeneous coordinates of the point.

**Conversion Between Color Model :-**

****

**Cohen-Sutherland Line Clippings:-**

****

There are 3 possibilities for the line −

Line can be completely inside the window (This line should be accepted).

Line can be completely outside of the window (This line will be completely removed from the region).

Line can be partially inside the window (We will find intersection point and draw only that portion of line that is inside region).

Algorithm

Step 1 : Assign a region code for two endpoints of given line.

Step 2 : If both endpoints have a region code 0000

then given line is completely inside.

Step 3 : Else, perform the logical AND operation for both region codes.

Step 3.1 : If the result is not 0000, then given line is completely

outside.

Step 3.2 : Else line is partially inside.

Step 3.2.1 : Choose an endpoint of the line

that is outside the given rectangle.

Step 3.2.2 : Find the intersection point of the

rectangular boundary (based 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

**The five main primitives in GKS are:**

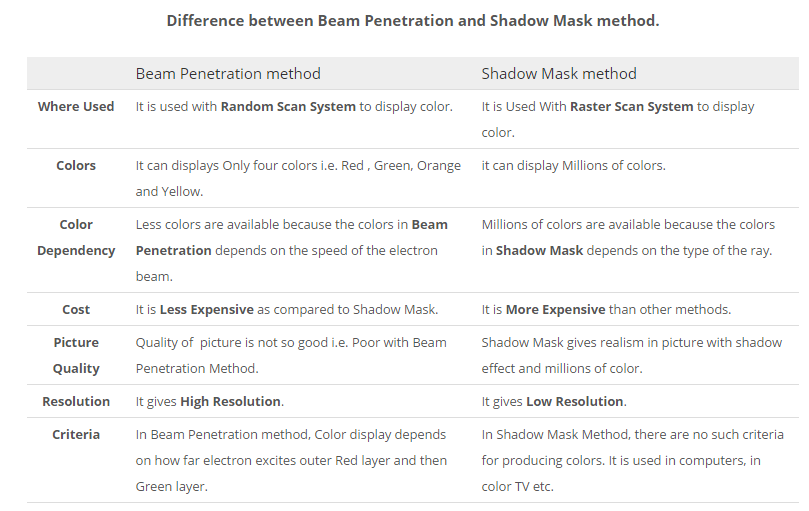
**polyline**: which draws a sequence of connected line segments.

**polymarker**: which marks a sequence of points with the same symbol.

**fill area:** which displays a specified area.

**text**: which draws a string of characters.

**cell array**: which displays an image composed of a variety of colours or grey scales.



**Raster Scan -**

In a raster scan system, the electron beam is swept across the screen, one row at a time from top to bottom. As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.

**Random Scan (Vector Scan) -**

In this technique, the electron beam is directed only to the part of the screen where the picture is to be drawn rather than scanning from left to right and top to bottom as in raster scan. It is also called vector display, stroke-writing display, or calligraphic display.

**Scan Conversion or Rasterization :-**

The process of representing continuous graphics object as a collection of discrete pixels is called Scan Conversion. For e.g a line is defined by its two end pts & the line equation, where as a circle is defined by its radius, center position & circle equation.

Examples of objects which can be scan converted:-

Point, Line, Sector, Arc Ellipse, Rectangle, Polygon, Characters, Filled Regions

**What is aliasing?**

In the line drawing algorithms, all rasterized locations do not match with the true

line and have to represent a straight line. This problem is severe in low resolution

screens. In such screens line appears like a stair-step. This effect is known as

aliasing.

**What is antialiasing?**

The process of adjusting intensities of the pixels along the line to minimize the

effect of aliasing .

**Types of antialiasing:-**

1.Supersampling 2.Area sampling 3. Pixel phasing

**DEPTH BUFFER or Z-BUFFER:-**

-Used for hidden surface detection

-Depth buffer is used to store depth values for (x, y) position, as surfaces are processed (0 ≤ depth ≤ 1).

Ref: <https://tutsmaster.org/advantages-and-disadvantages-of-z-buffer-algorithm/>

**BACK-FACE CULLING or BACK-FACE REMOVAL**

Back-face culling is a method in computer graphics programming which determines whether a polygon of a graphical object is visible. If not visible, the polygon is "culled" from rendering process, which increases efficiency by reducing the number of polygons that the hardware has to draw.

**FRAME BUFFER:-**

The frame buffer is used to store the intensity value of color value at each position (x, y).

**ENTROPY:-**

The entropy or average information of an image is a measure of the degree of randomness in the image. The entropy is useful in the context of image coding

Example:- Let us consider library A and library B with same number of books. Library A is properly arranged. B is not properly arranged. Then A has high entropy.

Library A has less entropy and more order. While B has higher entropy and less order. We need to spend energy to bring it to order.

So basically entropy is a measure of disorder or randomness. Higher the disorder higher the entropy

Ref comments in the video: <https://www.youtube.com/watch?v=YM-uykVfq_E&ab_channel=TED-Ed>

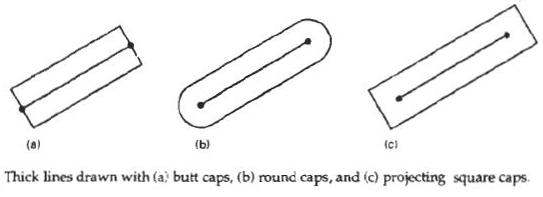
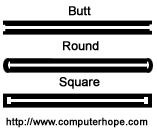
**What is a Line cap?**

Line caps can be used to adjust the shape of the line ends to give a better

appearance. There are three types of line caps. Butt cap which has a square end,

round cap which has a semi circle end, projecting square cap which has one half of

the line width beyond the specified end points.



**HSL stands** for hue, saturation, and lightness (or luminosity), and is also often called HLS.

**HSV stands** for hue, saturation, and value, and is also often called HSB (B for brightness).

A third model, common in computer vision applications, is **HSI**, for hue, saturation, and intensity.

Two techniques for producing **color in the CRT display** is

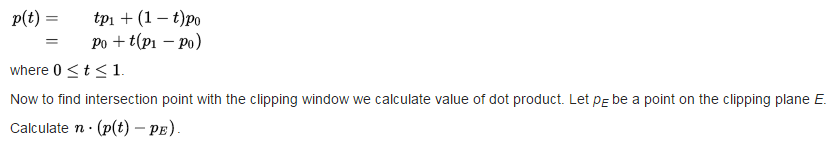
1. Beam Penetration method 2. Shadow mask method

A **vanishing point** is a point in the picture plane that is the intersection of the projections (or drawings) of a set of parallel lines in space on to the picture plane.



**Cyrus–Beck algorithm**is a generalized line clipping algorithm. uses repetitive clipping. convex polygon clipping window unlike Sutherland-Cohen that can be used only on a rectangular clipping area.

Here the parametric equation of a line in the view plane is:



Now to find intersection point with the clipping window we calculate value of dot product. Let pE be a point on the clipping plane E.

if> 0 vector pointed towards interior

if = 0 vector pointed parallel to plane containing p

if< 0 vector pointed away from interior

Here n stands for normal of the current clipping plane (pointed away from interior).

**Shading :-**

Shading is referred as the implementation of illumination model at the pixel points or polygon surfaces of the graphics objects .

**Phong Shading:-**

-normals are interpolated across the polygon

-need to know a normal for each vertex of the polygon

-better at dealing with highlights than Goraud shading

**Goraud Shading:-**

-colours are interpolated across the polygon

-need to know a normal for each vertex of the polygon

-computes the intensity calculations down and then across each scan lines thus eliminating the sharp edges.

**VECTOR (calligraphic) DISPLAYS:-**

lines drawn directly, no predefined grid

commands tell the electron gun where to move and when to turn on/off

+ lines are smooth

+ close to the 'pure mathematics' of the model

- slower with more elements to be drawn, can start to flicker

- only lines possible, no filled polygons, or bitmaps

- monochrome for each electron gun

**RASTER DISPLAYS:-**

image represented by a rectangular grid of pixels (picture elements)

image stored in a frame buffer

electron gun(s) continually scanning in a regular pattern (line by line across entire screen)

+ constant time to redraw any number of elements

+ no flicker

- jagged edges from conversion to pixels

- discretized version of the model

**IMAGE COMPRESSION**

There are two general types of compression algorithms:

1. Lossless compression

2. Lossy compression

**Lossless Compression Algorithms**

The various algorithms used to implement lossless data compression are :

1. Run length encoding

2. Differential pulse code modulation

3. Dictionary based encoding

4. Huffman Coding algorithm

5. LZW

**Lossy Data Compression Algorithm**

1. Discrete cosine transform
2. Discrete Wavelet transform

**1. Run length encoding**

• This method replaces the consecutive occurrences of a given symbol with only one copy of the symbol along with a count of how many times that symbol occurs. Hence the names ‘run length'.

• For example, the string AAABBCDDDD would be encoded as 3A2BIC4D.

**3. Dictionary based encoding**

• One of the best known dictionary based encoding algorithms is Lempel-Ziv (LZ) compression algorithm.

• This method is also known as substitution coder.

• In this method, a dictionary (table) of variable length strings (common phrases) is built.

**IMAGE PROCESSING**

**BLIND DECONVOLUTION:-**

blind deconvolution is a deconvolution technique that permits recovery of the target scene from a single or set of "blurred" images in the presence of a poorly determined or unknown point spread function (PSF)