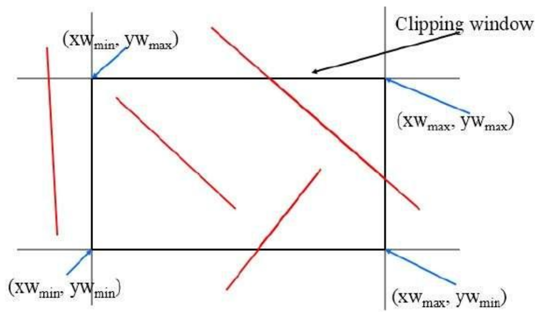
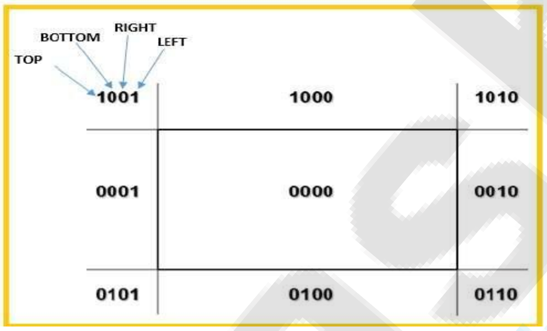
Cohen Sutherland Line clipping algorithm

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.

Algorithm

Step 1 − Assign a region code for each endpoint.

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

Midpoint line clipping 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.

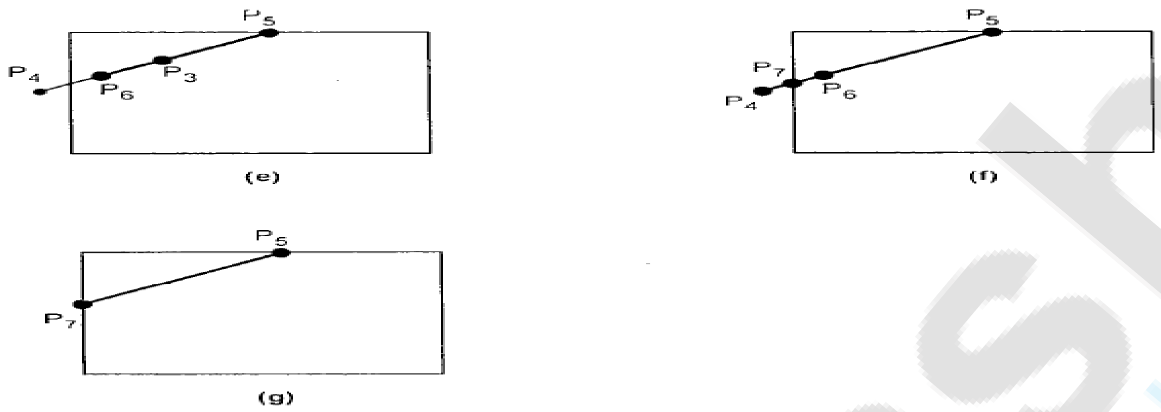
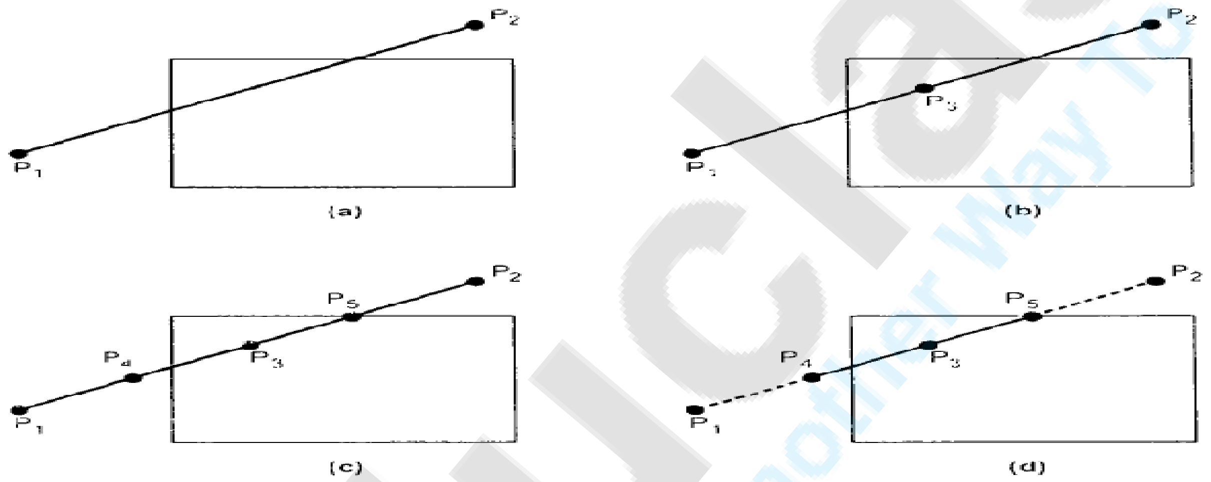


Fig. (k) Clipping line with midpoint subdivision algorithm



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 ($x > Wx_2$)

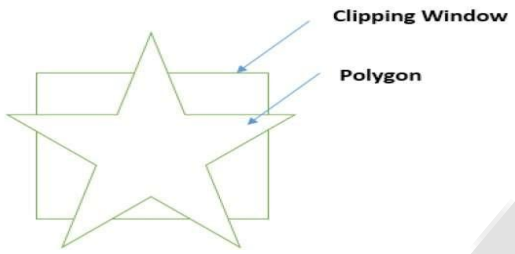
   Set Bit 3 - if ($y < Wy_1$)

   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.

Polygon clipping algorithm

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 –

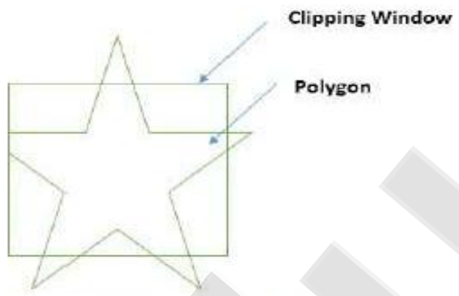


Figure: Clipping Left Edge

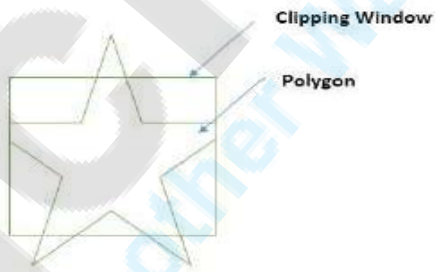


Figure: Clipping Right Edge

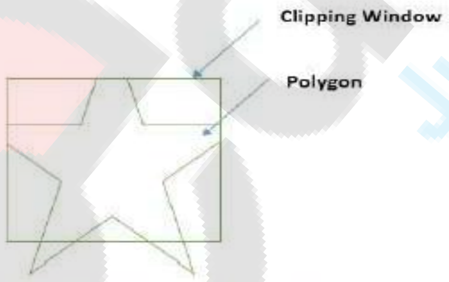


Figure: Clipping Top Edge

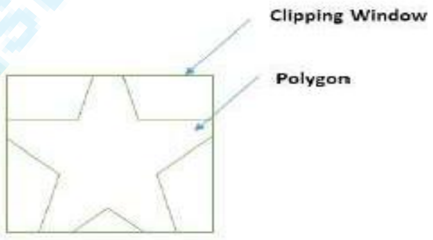


Figure: Clipping Bottom Edge

3D Translation

- Translation can best be described as linear change in position. This change can be represented by a delta vector [dx, dy, dz], where dx represents the change in the object's x position, dy represents the change in its y position, and dz its z position.

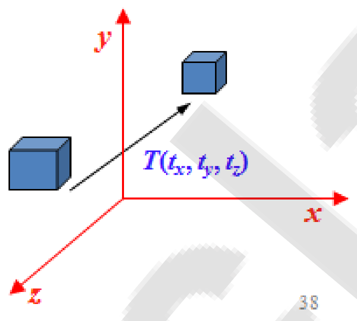
- Translation is defined by a delta vector *v* = (*Tx, Ty, Tz*) and by a transformation matrix

- Repositioning an object along a straight line path from one co-ordinate location to another
- (*x,y,z*) --→ (*x′,y′,z′*)
- To translate a 3D position, we add translation distances $t_x t_y$ and t_z to the original coordinates (*x,y,z*) to obtain the new coordinate position (*x′,y′*)
- $x' = x + t_x$, $y' = y + t_y$, $z' = z + t_z$

Matrix form (4 × 4)

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

$$P' = T(t_x, t_y, t_z)P$$



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

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

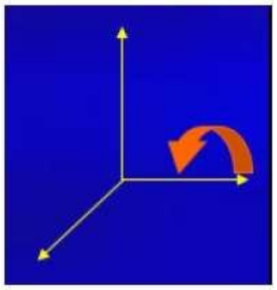
3D Rotation.

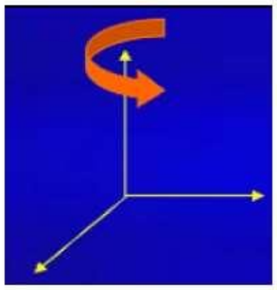
3D rotation is not same as 2D rotation. In 3D rotation, we have to specify the angle of rotation along with the axis of rotation. We can perform 3D rotation about X, Y, and Z axes. They are represented in the matrix form as below –

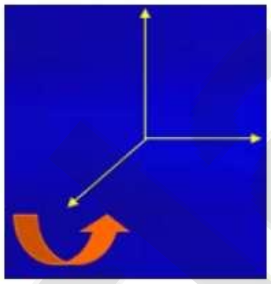
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_z(\theta)$$

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

The following figure explains the rotation ab


Rotation about x-axis


Rotation about y-axis


Rotation about z-axis

out various axes –

3D Scaling

We scale a 3D object with respect to the origin by setting the scaling factors s_x, s_y and s_z, which are multiplied to the original vertex coordinate positions (x,y,z):

$$x' = x * s_x,\ y' = y * s_y,\ z' = z * s_z$$

Alternatively, this scaling can also be specified by the transformation matrix in the following formula:

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

Exercise: Scale a triangle with vertices at original coordinates (10,25,5), (5,10,5), (20,10,10) by s_x=1.5, s_y=2, and s_z=0.5 with respect to the origin. For verification, roughly plot the x and y values of the original and resultant triangles, and imagine the locations of z values.

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

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

$z2=z1 \cdot sz$

Vector (x, y, z, 1) is to be magnified to (x*Sx, y*Sy, z*Sz, 1)

The vector will be multiplied by the transformation matrix :

```
| sx 0  0  0 |
|  x y  z  1 | * | 0 sy 0 0 | = |x*sx y*sy z*sz 1|
|  0 0 sz  0 |
|  0 0  0  1 |
```

You can change the size of an object using scaling transformation. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.

3D Reflection

Reflection is accomplished in a ray trace renderer by following a ray from the eye to the mirror and then calculating where it bounces from, and continuing the process until no surface is found, or a non-reflective surface is found. Reflection on a shiny surface like wood or tile can add to the photorealistic effects of a 3D rendering.

- **Polished** - A polished reflection is an undisturbed reflection, like a mirror or chrome.
- **Blurry** - A blurry reflection means that tiny random bumps on the surface of the material cause the reflection to be blurry.
- **Metallic** - A reflection is metallic if the highlights and reflections retain the color of the reflective object.
- **Glossy** - This term can be misused. Sometimes, it is a setting which is the opposite of blurry (e.g. when "glossiness" has a low value, the reflection is blurry). However, some people use the term "glossy reflection" as a synonym for "blurred reflection". Glossy used in this context means that the reflection is actually blurred.
  - Transformation matrix for a reflection through *X-Y plane* is:
  -

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

  -
  - Transformation matrix for a reflection through *Y-Z plane* is:

- $$T_{yz} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
-
- Transformation matrix for a reflection through *Z-X plane* is:
- $$T_{yz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Shearing

A transformation that slants the shape of an object is called the **shear transformation**. Like in 2D shear, we can shear an object along the X-axis, Y-axis, or Z-axis in 3D.

As shown in the above figure, there is a coordinate P. You can shear it to get a new coordinate P', which can be represented in 3D matrix form as below −

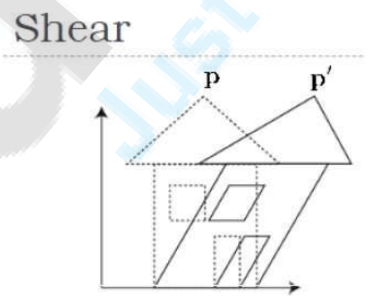
$$Sh = \begin{bmatrix} 1 & sh_x^y & sh_x^z & 0 \\ sh_y^x & 1 & sh_y^z & 0 \\ sh_z^x & sh_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

P′ = P · Sh

$$X' = X + Sh_x^y Y + Sh_x^z Z$$

$$Y' = Sh_y^x X + Y + sh_y^z Z$$

$$Z' = Sh_z^x X + Sh_z^y Y + Z$$



In space, we divide shear transformation according to the direction of the surfaces xy,xz and yz. Values of Sx,Sy and Sz determine shear transformation sizes for all the directions.
A shear transformation about the xy plane

```
     | 1 0 0 0 |
     | 0 1 0 0 |
Axy = | Sx Sy 0 0 |.
```

```
     |  0  0 0 1 |
A shear matrix about the xz plane
     |  1  0  0  0 |
     | Sx  1 Sz  0 |
Axz = |  0  1  1  0 |.
     |  0  0  0  1 |
A shear matrix about the yz plane
     | 1 Sy Sz 0 |
     | 0  1  0 0 |
Ayz = | 0  0  1 0 |.
     | 0  0  0 1 |
```

Fractal

A French/American mathematician Dr Benoit Mandelbrot discovered Fractals. The word fractal was derived from a Latin word *fractus* which means broken.

What are Fractals?

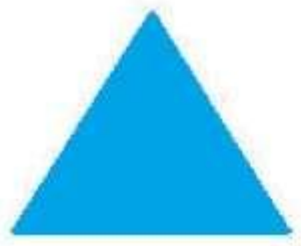
Fractals are very complex pictures generated by a computer from a single formula. They are created using iterations. This means one formula is repeated with slightly different values over and over again, taking into account the results from the previous iteration.

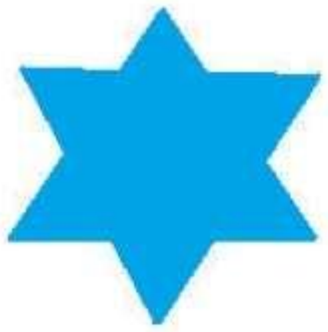
Fractals are used in many areas such as −
Astronomy − For analyzing galaxies, rings of Saturn, etc.
Biology/Chemistry − For depicting bacteria cultures, Chemical reactions, human anatomy, molecules, plants,
Others − For depicting clouds, coastline and borderlines, data compression, diffusion, economy, fractal art, fractal music, landscapes, special effect, etc.

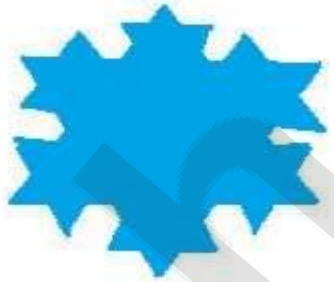
Generation of Fractals

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.


(a) Zeroth Generation


(b) First Generation


(c) Second Generation

Van Koch Curve

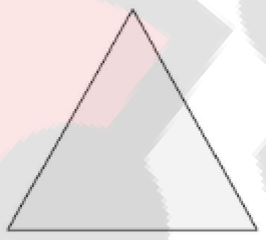
The Koch snowflake (also known as the Koch curve, Koch star, or Koch island) is a mathematical curve and one of the earliest fractal curves to have been described. It is based on the Koch curve, which appeared in a 1904 paper titled "On a continuous curve without tangents, constructible from elementary 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.

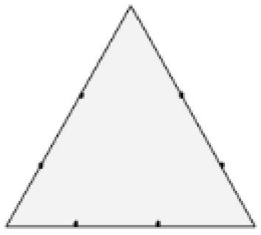
Construction

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.

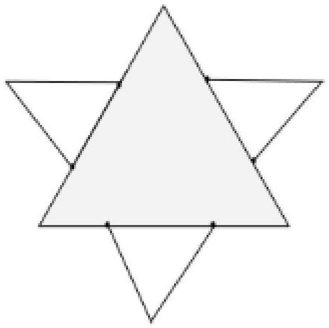
It's best if the length of the sides are divisible by 3, because of the nature of this fractal. This will become clear in the next few steps.



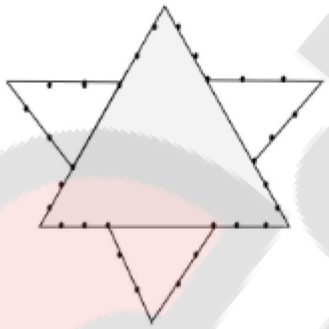
Step2:
Divide each side in three equal parts. This is why it is handy to have the sides divisible by three.



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.

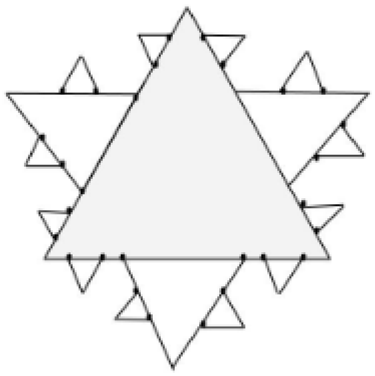


Step4:
Divide each outer side into thirds. You can see the 2nd generation of triangles covers a bit of the first. These three line segments shouldn't be parted in three.



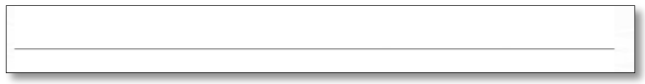
Step5:
Draw an equilateral triangle on each middle part.

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



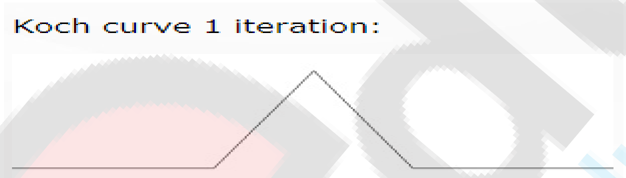
Koch Snowflake

The Koch curve fractal was first introduced in 1904 by Helge von Koch. It was one of the first fractal objects to be described.



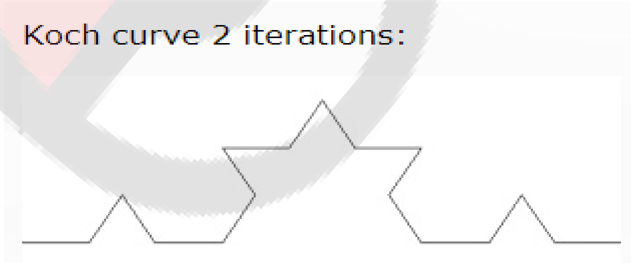
To create a Koch curve

- create a line and divide it into three parts
- the second part is now rotated by 60°
- add another part which goes from the end of part 2 to to the beginning of part 3
- repeat with each part

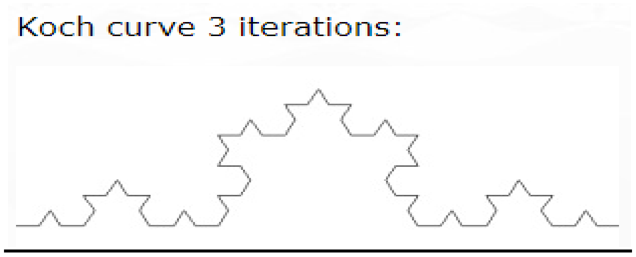


Mathematical aspects:

The perimeter of the Koch curve is increased by 1/4. That implys that the perimeter after an infinite number of iterations is infinite. The formula for the perimeter after k iterations is:



The number of the lines in a Koch curve can be determined with following formula:

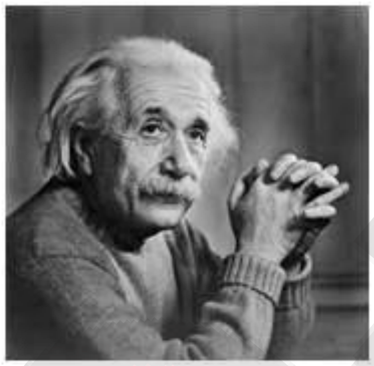


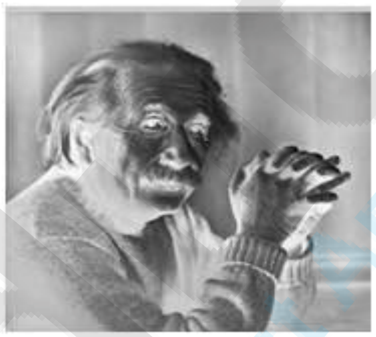
Negative transformation

The second linear transformation is negative transformation, which is invert of identity transformation. In negative transformation, each value of the input image is subtracted from the L-1 and mapped onto the output image.

The result is somewhat like this.

Input Image   Output Image





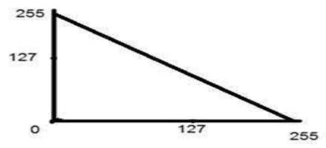
In this case the following transition has been done.

$s = (L - 1) - r$

since the input image of Einstein is an 8 bpp image, so the number of levels in this image are 256. Putting 256 in the equation, we get this

$s = 255 - r$

So each value is subtracted by 255 and the result image has been shown above. So what happens is that, the lighter pixels become dark and the darker picture becomes light. And it results in image negative.
It has been shown in the graph below.



Logarithmic transformation.

Logarithmic transformation further contains two type of transformation. Log transformation and inverse log transformation.

Log transformation

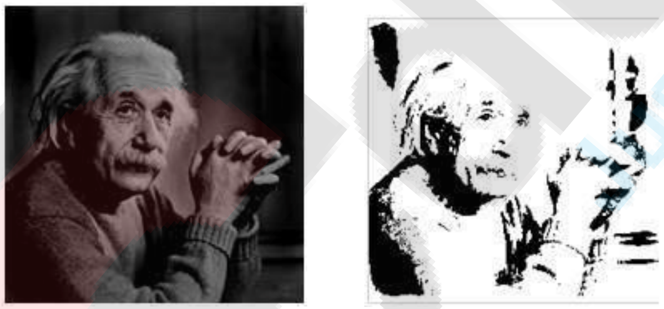
The log transformations can be defined by this formula

$s = c \log(r + 1)$.

Where s and r are the pixel values of the output and the input image and c is a constant. The value 1 is added to each of the pixel value of the input image because if there is a pixel intensity of 0 in the image, then log (0) is equal to infinity. So 1 is added, to make the minimum value at least 1.

During log transformation, the dark pixels in an image are expanded as compare to the higher pixel values. The higher pixel values are kind of compressed in log transformation. This result in following image enhancement.

The value of c in the log transform adjust the kind of enhancement you are looking for.



The inverse log transform is opposite to log transform.

Power-Law transformation.

There are further two transformation is power law transformations, that include nth power and nth root transformation. These transformations can be given by the expression:

$S = c\, r \wedge \gamma$

This symbol γ is called gamma, due to which this transformation is also known as gamma transformation.

Variation in the value of γ varies the enhancement of the images. Different display devices / monitors have their own gamma correction, that's why they display their image at different intensity.

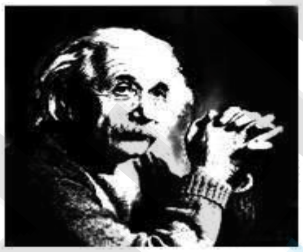
This type of transformation is used for enhancing images for different type of display devices. The gamma of different display devices is different. For example Gamma of CRT lies in between of 1.8 to 2.5, that means the image displayed on CRT is dark.

Correcting gamma.

$S = c r ^\wedge \gamma$
$S = cr ^\wedge (1/2.5)$
The same image but with different gamma values has been shown here.

For example

Gamma = 10



Contrast Stretching transformation.

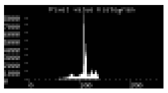
The contrast of an image is a measure of its dynamic range, or the "spread" of its histogram. The dynamic range of an image is defined to be the entire range of intensity values contained within an image, or put a simpler way, the maximum pixel value minus the minimum pixel value. An 8-bit image at full contrast has a dynamic range of 28-l=255, with anything less than that yielding a lower contrast image.

The minimum pixel value in Figure 3-4a is 81, and the maximum pixel value is 127. Increasing the dynamic range entails "stretching" the histogram in Figure 3-4b by applying a linear scaling function that maps pixel values of 81 to 0, maps pixel values of 127 to 255, and scales pixel intensities that lie within the range [82-126] accordingly.

Normalization is commonly used to improve the contrast in an image without distorting relative graylevel intensities too significantly.

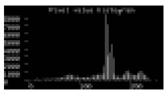
We begin by considering an image

which can easily be enhanced by the most simple of contrast stretching implementations because the intensity histogram forms a tight, narrow cluster between the graylevel intensity values of 79 - 136, as shown in



After contrast stretching, using a simple linear interpolation between *c = 79* and *d = 136*, we obtain

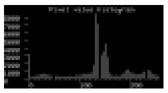
Compare the histogram of the original image with that of the contrast-stretched version



While this result is a significant improvement over the original, the enhanced image itself still appears somewhat flat. Histogram equalizing the image increases contrast dramatically, but yields an artificial-looking result

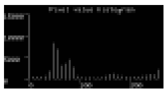
In this case, we can achieve better results by contrast stretching the image over a more narrow range of graylevel values from the original image. For example, by setting the cutoff fraction parameter to 0.03, we obtain the contrast-stretched image

and its corresponding histogram



Note that this operation has effectively spread out the information contained in the original histogram peak (thus improving contrast in the interesting face regions) by pushing those intensity levels to the left of the peak down the histogram *x*-axis towards 0. Setting the cutoff fraction to a higher value, *e.g.* 0.125, yields the contrast stretched image

As shown in the histogram



most of the information to the left of the peak in the original image is mapped to 0 so that the peak can spread out even further and begin pushing values to its right up to 255.

Bit plane slicing

Instead of highlighting gray level images, highlighting the contribution made to total image appearance by specific bits might be desired. Suppose that each pixel in an image is represented by 8 bits. Imagine the image is composed of 8, 1-bit planes ranging from bit plane1-0 (LSB)to bit plane 7 (MSB).

In terms of 8-bits bytes, plane 0 contains all lowest order bits in the bytes comprising the pixels in the image and plane 7 contains all high order bits.

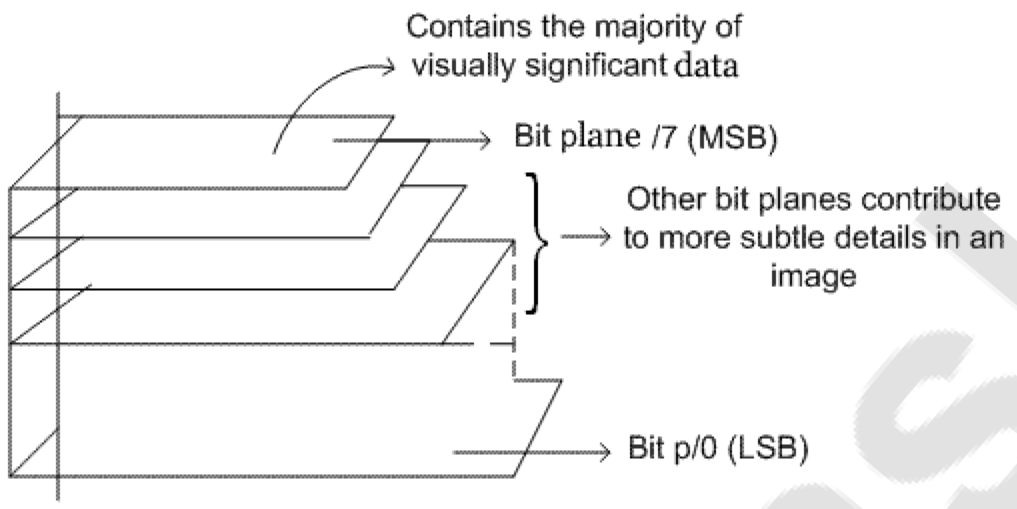

Contains the majority of visually significant data
Bit plane /7 (MSB)
Other bit planes contribute to more subtle details in an image
Bit p/0 (LSB)

Figure (5.15)

Separating a digital image into its bit planes is useful for analyzing the relative importance played by each bit of the image, implying, it determines the adequacy of numbers of bits used to quantize each pixel , useful for image compression.

In terms of bit-plane extraction for a 8-bit image, it is seen that binary image for bit plane 7 is obtained by proceeding the input image with a thresholding gray-level transformation function that maps all levels between 0 and 127 to one level (e.g 0)and maps all levels from 129 to 253 to another (eg. 255).

Gray level slicing with Background

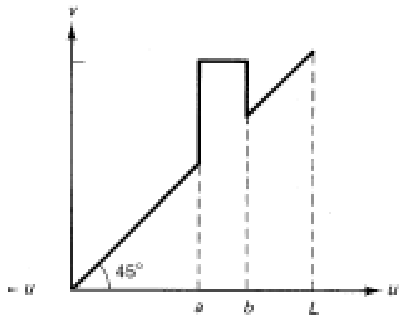
Lillesand and Kiefer (1994) explained the goal of image enhancement procedures is to improve the visual interpretability of any image by increasing the apparent distinction between the features in the scene.

This objective is to create "new" image from the original image in order to increase the amount of information that can be visually interpreted from the data.

Enhancement operations are normally applied to image data after the appropriate restoration procedures have been performed. Noise removal, in particular, is an important precursor to most enhancements. In this study, typical image enhancement techniques are as follows:

Grey level thresholding

Grey level thresholding is a simple lookup table, which partitions the gray levels in an image into one or two categories - those below a user-selected threshold and those above. Thresholding is one of many methods for creating a binary mask for an image. Such masks are used to restrict subsequent processing to a particular region within an image.



Level slicing
Level slicing is an enhancement technique whereby the Digital Numbers (DN) distributed along the x-axis of an image histogram is divided into a series of analyst-specified intervals of "slices". All of DNs falling within a given interval in the input image are then displayed at a single DN in the output image (Lillesand and Kiefer, 1994).

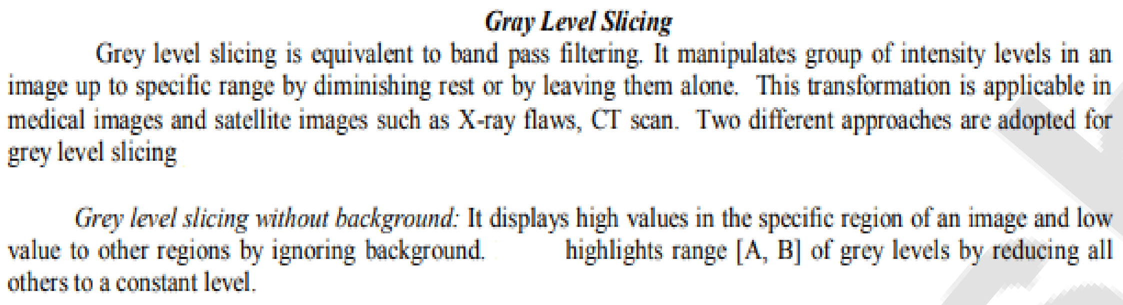
Contrast stretching

Most satellites and airborne sensor were designed to accommodate a wide range of illumination conditions, from poorly lit arctic regions to high reflectance desert regions.
Because of this, the pixel values in the majority of digital scenes occupy a relatively small portion of the possible range of image values.
If the pixel values are displayed in their original form, only a small range of gray values will be used, resulting in a low contrast display on which similar features night is indistinguishable.

A contrast stretch enhancement expands the range of pixel values so that they are displayed over a fuller range of gray values. (PCI, 1997)

Generally, image display and recording devices typically operate over a range of 256 gray levels (the maximum number represent in 8-bit computer encoding).
In the case of 8-bit single image, is to expand the narrow range of brightness values typically present in an output image over a wider range of gray value.
The result is an output image that is designed to accentuate the contrast between features of interest to the image analyst (Lillesand and Kiefer, 1994).

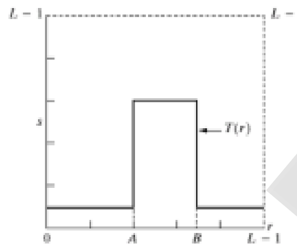
Gray level slicing without Background



Gray Level Slicing

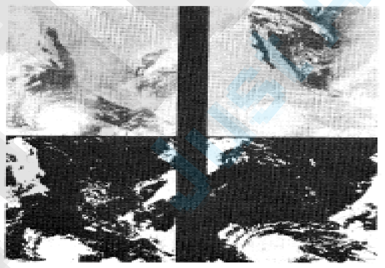
Grey level slicing is equivalent to band pass filtering. It manipulates group of intensity levels in an image up to specific range by diminishing rest or by leaving them alone. This transformation is applicable in medical images and satellite images such as X-ray flaws, CT scan. Two different approaches are adopted for grey level slicing

Grey level slicing without background: It displays high values in the specific region of an image and low value to other regions by ignoring background. highlights range [A, B] of grey levels by reducing all others to a constant level.



Range [A, B] of grey levels by reducing all others to a constant level.

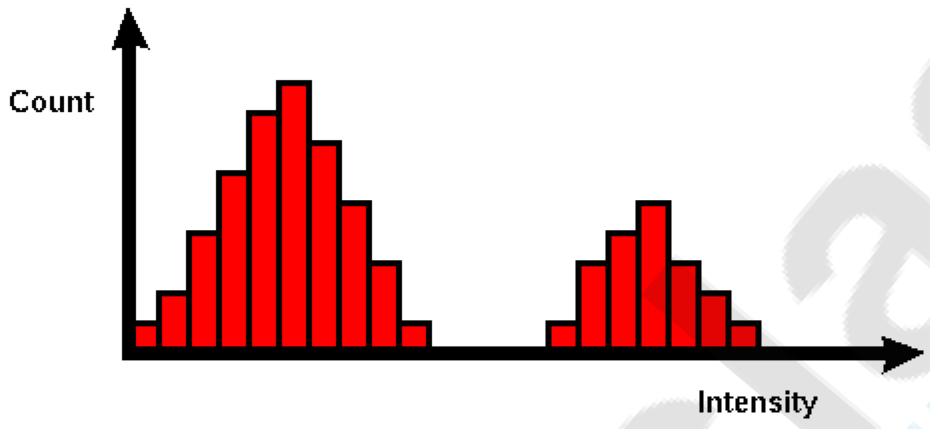
These transformations permit segmentation of certain gray level regions from the rest of the image. This technique is useful when different features of an image are contained in different gray levels. The following image shows the result of intensity window slicing for segmentation of low-temperature regions (clouds) of two images where high intensity gray levels are proportional to low temperatures.



Level slicing of intensity window *[175,250]*.
Top row: visual and infrared images, bottom: segmented image

Histogram

In an image processing context, the histogram of an image normally refers to a histogram of the pixel intensity values. This histogram is a graph showing the number of pixels in an image at each different intensity value found in that image. For an 8-bit grayscale image there are 256 different possible intensities, and so the histogram will graphically display 256 numbers showing the distribution of pixels amongst those grayscale values. Histograms can also be taken of color images --- either individual histograms of red, green and blue channels can be taken, or a 3-D histogram can be produced, with the three axes representing the red, blue and green channels, and brightness at each point representing the pixel count. The exact output from the operation depends upon the implementation --- it may simply be a picture of the required histogram in a suitable image format, or it may be a data file of some sort representing the histogram statistics.



Common Names: Histogram

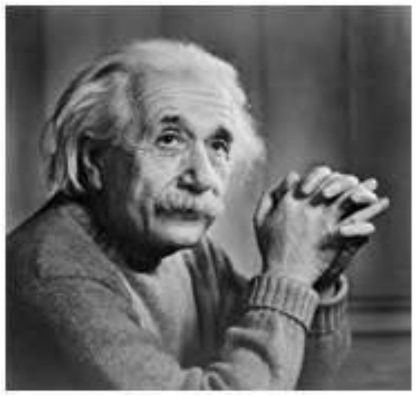
How It Works

The operation is very simple. The image is scanned in a single pass and a running count of the number of pixels found at each intensity value is kept. This is then used to construct a suitable histogram.

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.



Histogram of this image

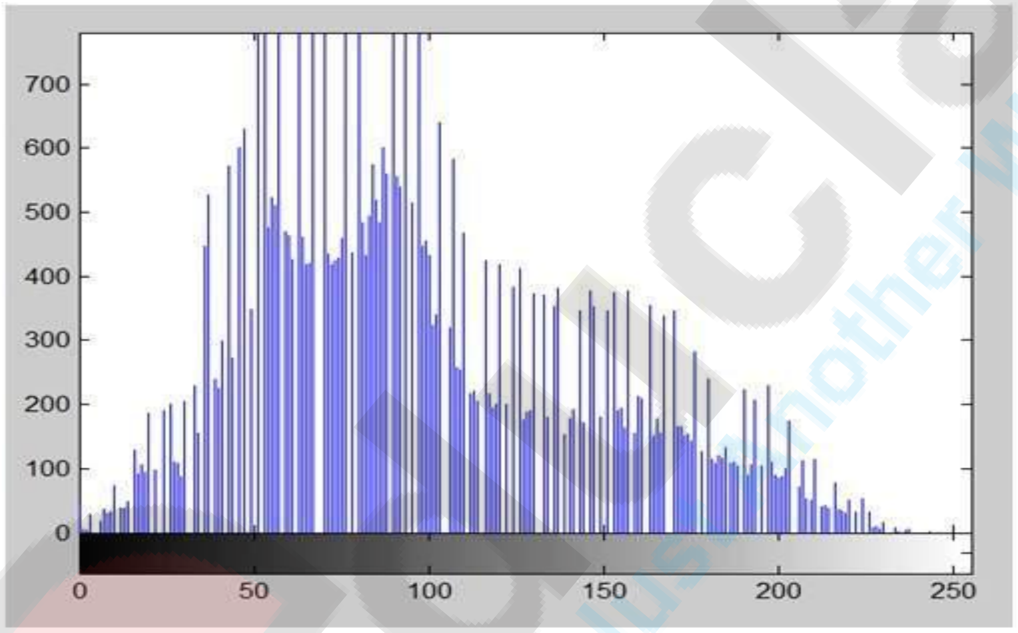
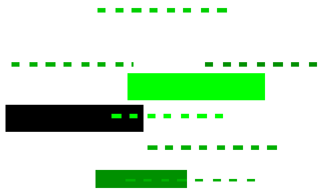


Image Subtraction

Image subtraction or **pixel subtraction** is a process whereby the digital numeric value of one pixel or whole image is subtracted from another image. This is primarily done for one of two reasons – levelling uneven sections of an image such as half an image having a shadow on it, or detecting changes between two images.[1] This detection of changes can be used to tell if something in the image moved.



Common Names: Pixel difference, Pixel subtract

The pixel subtraction operator takes two images as input and produces as output a third image whose pixel values are simply those of the first image minus the corresponding pixel values from the second image. It is also often possible to just use a single image as input and subtract a constant value from all the pixels. Some versions of the operator will just output the absolute difference between pixel values, rather than the straightforward signed output.

How It Works

The subtraction of two images is performed straightforwardly in a single pass. The output pixel values are given by:

$$Q(i,j) = P_1(i,j) - P_2(i,j)$$

Image Averaging

Image averaging is obtained by finding the average of K images.

It is applied in de-noising the images.

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

A noisy image is defined by:

$$g(x,y) = f(x,y) + \eta(x,y)$$

Assuming that the noise is uncorrelated with zero mean