Digital Image Processing Lecture 2

Muhammad Usman Ghani Khan

Department of Computer Science

Autumn Semester

Outline

Agenda

Today Discussion

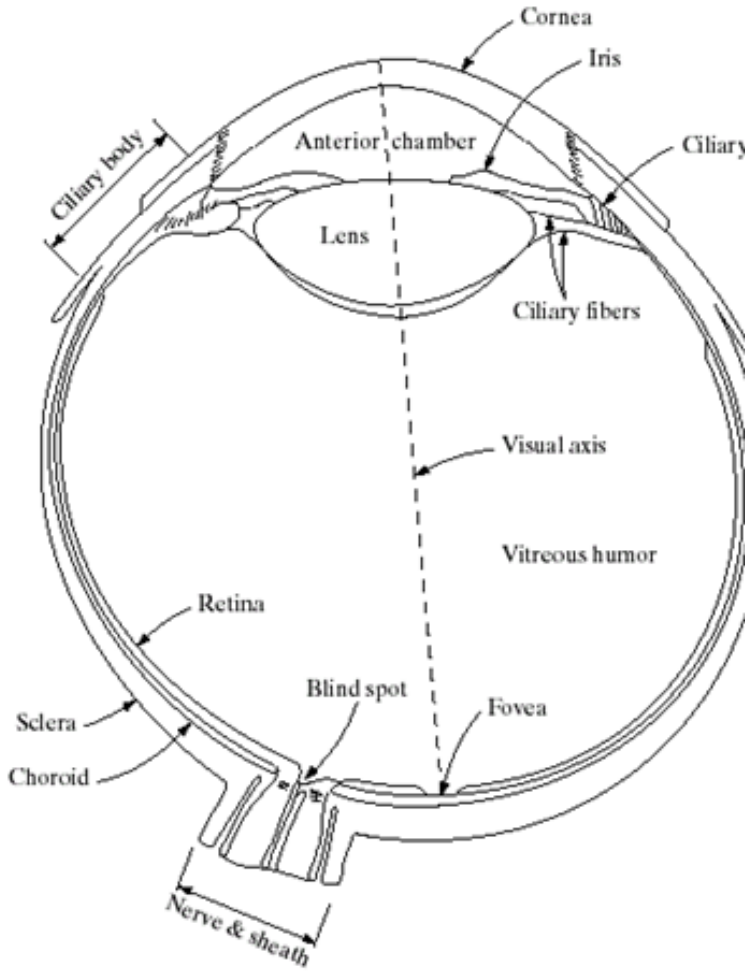
- The human visual system
- Light and the electromagnetic spectrum
- Image representation
- Image sensing and acquisition
- Sampling, quantisation and resolution

Human Visual System

- The best vision model we have!
- Knowledge of how images form in the eye can help us with processing digital images
- We will take just a whirlwind tour of the human visual system

Structure Of The Human Eye

- The lens focuses light from objects onto the retina
- The retina is covered with light receptors called cones (6-7 million) and rods (75-150 million)
- Cones are concentrated around the fovea and are very sensitive to colour
- Rods are more spread out and are sensitive to low levels of illumination
- Cornea and sclera keep everything in!
- Choroid contains all of the blood vessels that serve as nutrition to the eye
- Iris contains the pigment that gives our eyes their colour
- Rods for general vision while cones for details
- Blind spot experiment: Blind spot is where the optic nerve connects
- Women have extra cones which means they can see more colours



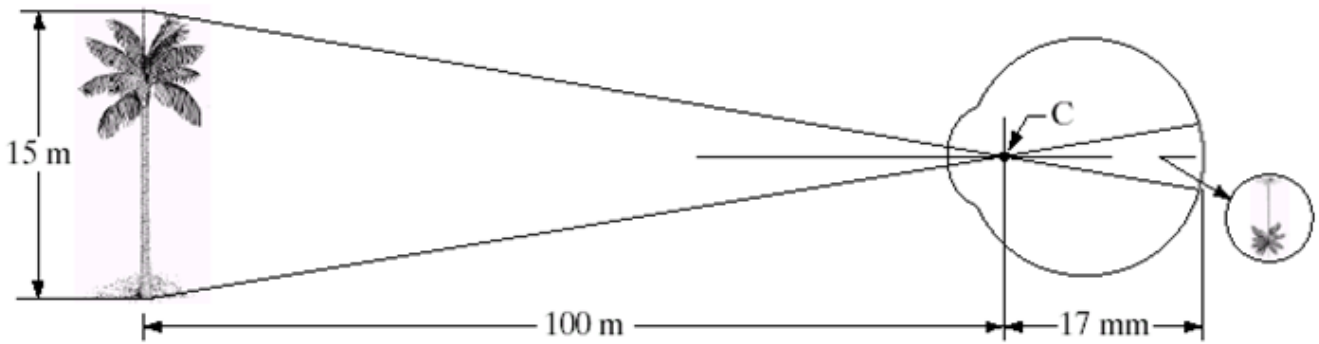
Blind-Spot Experiment

Draw an image similar to that below on a piece of paper (the dot and cross are about 6 inches apart)

- Close your right eye and focus on the cross with your left eye
- Hold the image about 20 inches away from your face and move it slowly towards you
- The dot should disappear!

Image Formation In The Eye

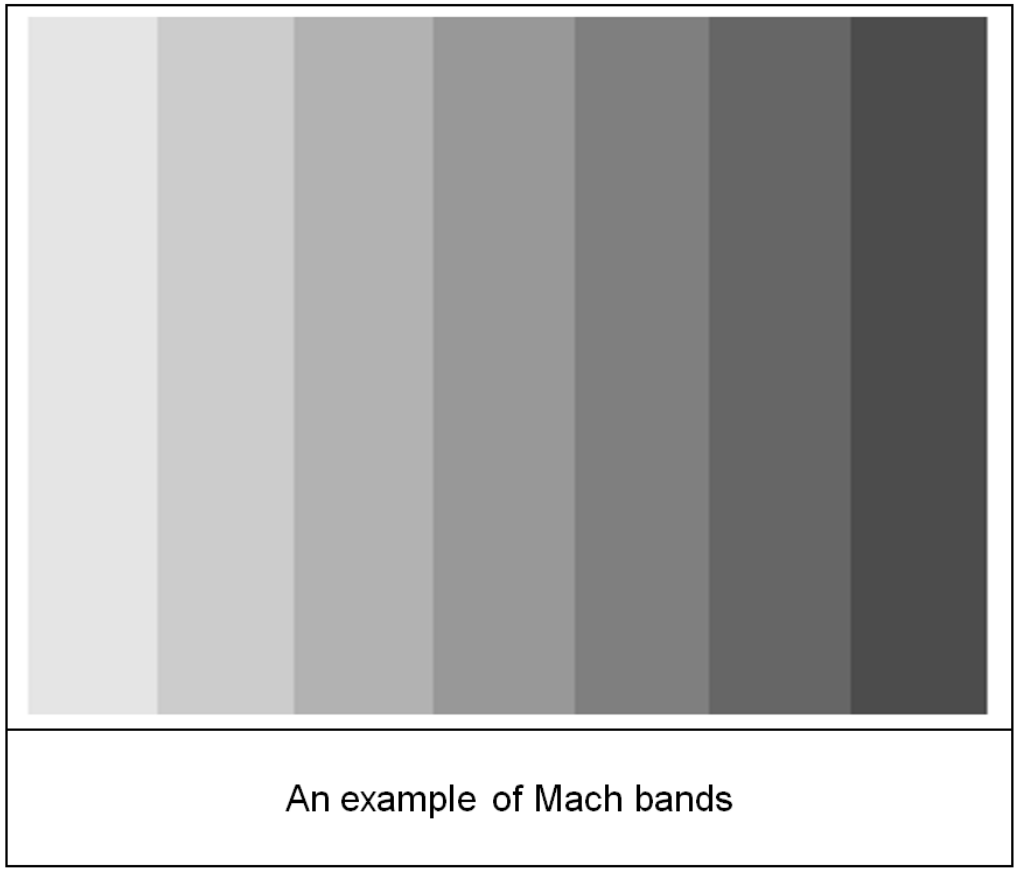
- Muscles within the eye can be used to change the shape of the lens allowing us focus on objects that are near or far away
- An image is focused onto the retina causing rods and cones to become excited which ultimately send signals to the brain



Brightness Adaptation & Discrimination

- The human visual system can perceive approximately 10^{10} different light intensity levels
- However, at any one time we can only discriminate between a much smaller number — brightness adaptation
- Similarly, the perceived intensity of a region is related to the light intensities of the regions surrounding it

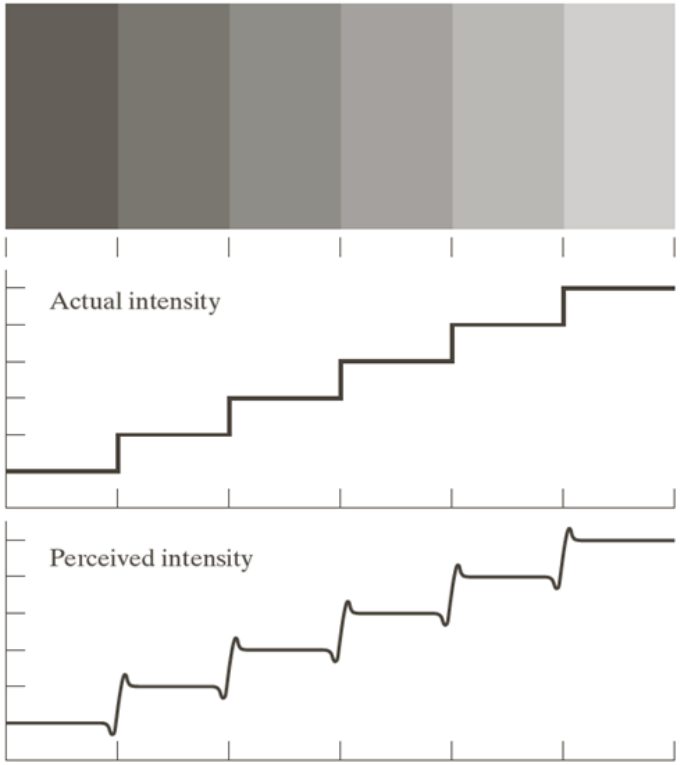
Brightness Adaptation & Discrimination



An example of Mach bands

Figure: Mach bands were named after Ernst Mach who described the phenomenon in 1865

Human Eye Perception

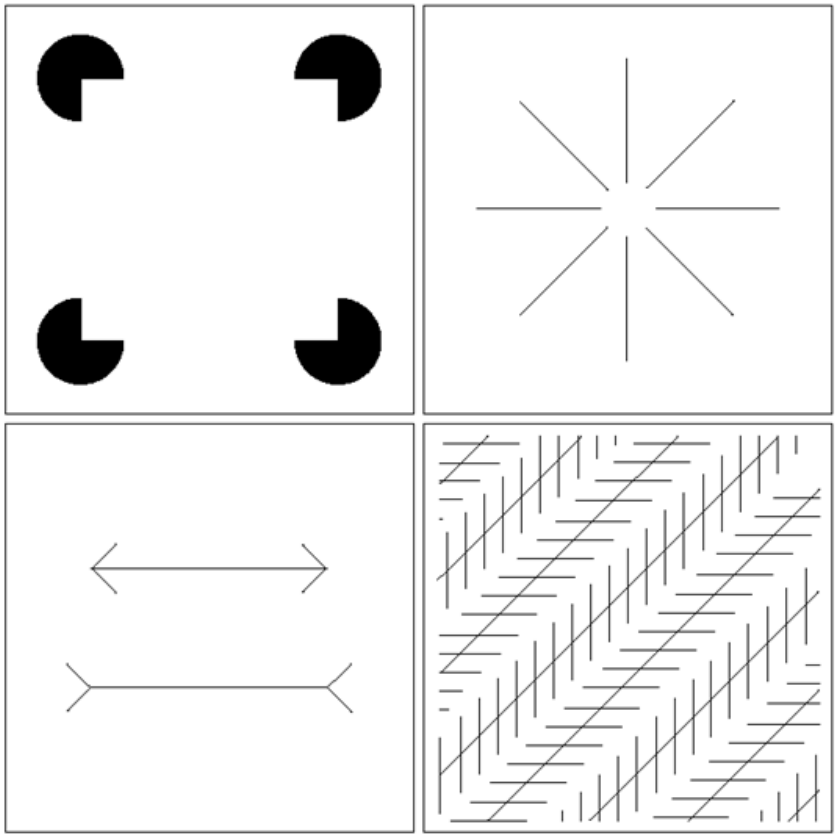


Simultaneous Contrast

An example of *simultaneous contrast*

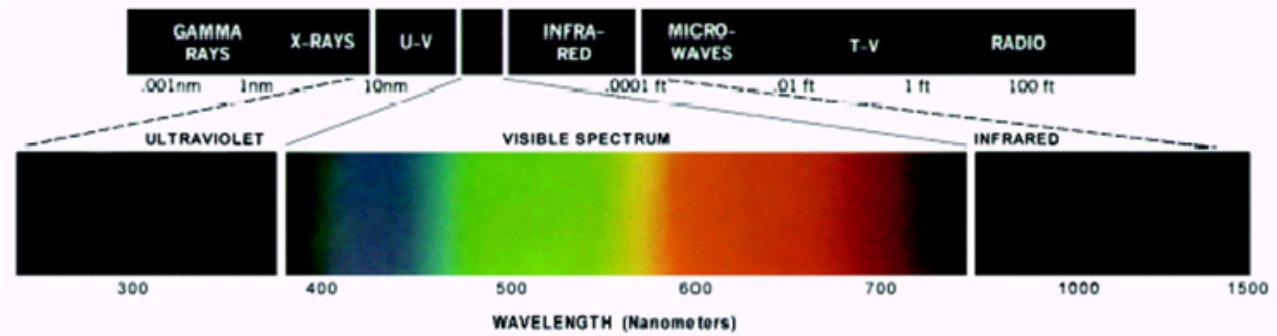
Optical Illusions

Our visual systems play lots of interesting tricks on us



Light And The Electromagnetic Spectrum

- Light is just a particular part of the electromagnetic spectrum that can be sensed by the human eye
- The electromagnetic spectrum is split up according to the wavelengths of different forms of energy the brain
- In 1666 Sir Isaac Newton discovered that light passed through a prism splits into a continuous spectrum of colours
- Many image applications use electromagnetic radiation that is far outside the visual spectrum, *i.e.*, x-ray images, infra-red images; we generate images of these images by mapping them to the visual spectrum



Reflected Light

- The colours that we perceive are determined by the nature of the light reflected from an object
- For example, if white light is shone onto a green object most wavelengths are absorbed, while green light is reflected from the object

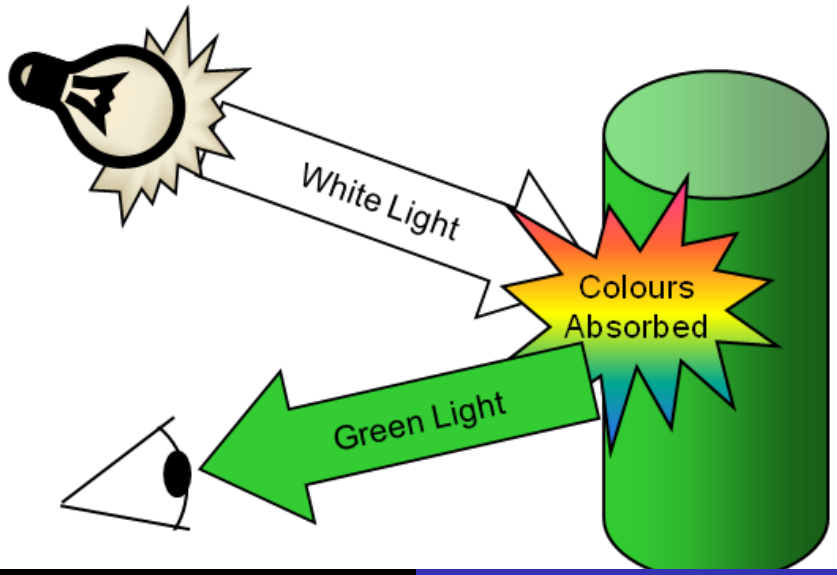


Image representation

- Before we discuss image acquisition recall that a digital image is composed of M rows and N columns of pixels each storing a value
- Pixel values are most often grey levels in the range 0-255(black-white)
- We will see later on that images can easily be represented as matrices

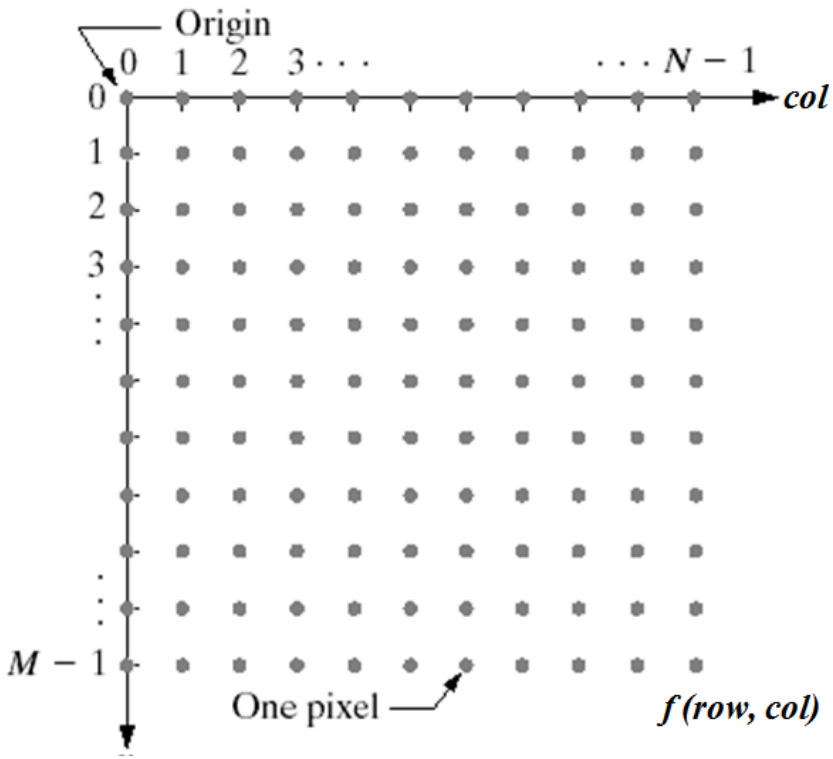
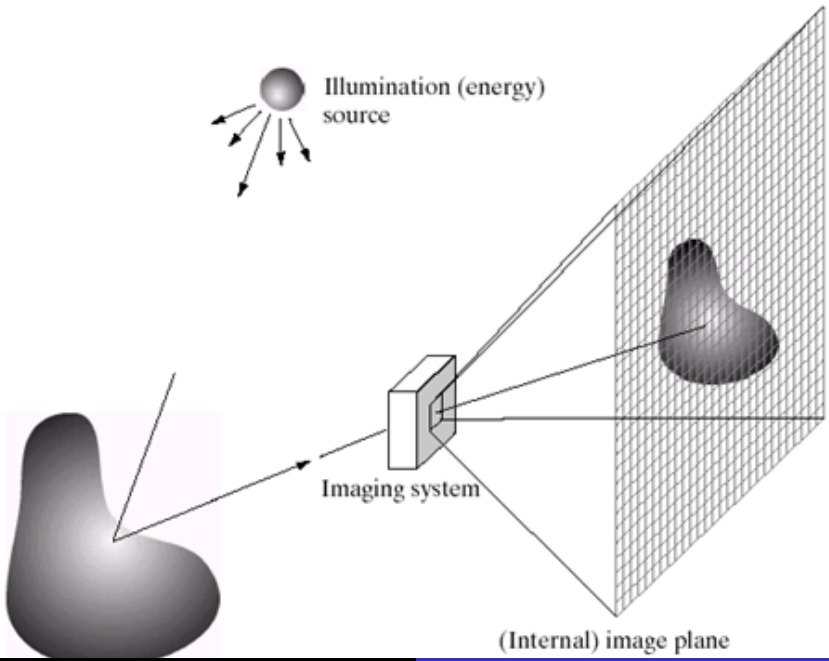


Image Acquisition

- Images are typically generated by illuminating a scene and absorbing the energy reflected by the objects in that scene
- Typical notions of illumination and scene can be way off:
  - X-rays of a skeleton
  - Ultrasound of an unborn baby
  - Electro-microscopic images of molecules



Conversion of analog signal to digital signal

- Why we need digital signal?
- sampling
- quantization
- convert both of its axis (x,y) into a digital format.

Image Sampling And Quantisation

- A digital sensor can only measure a limited number of samples at a discrete set of energy levels
- Quantisation is the process of converting a continuous analogue signal into a digital representation of this signal

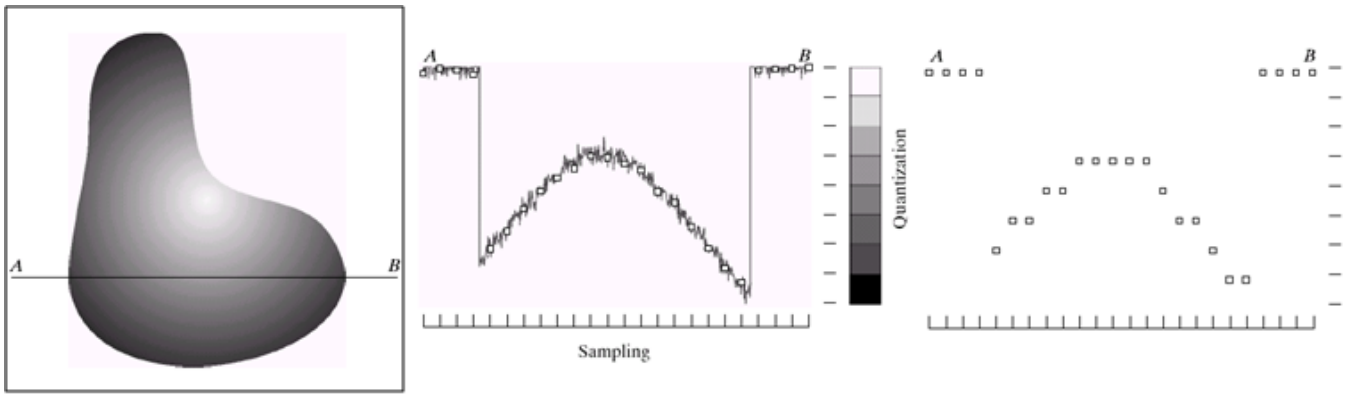
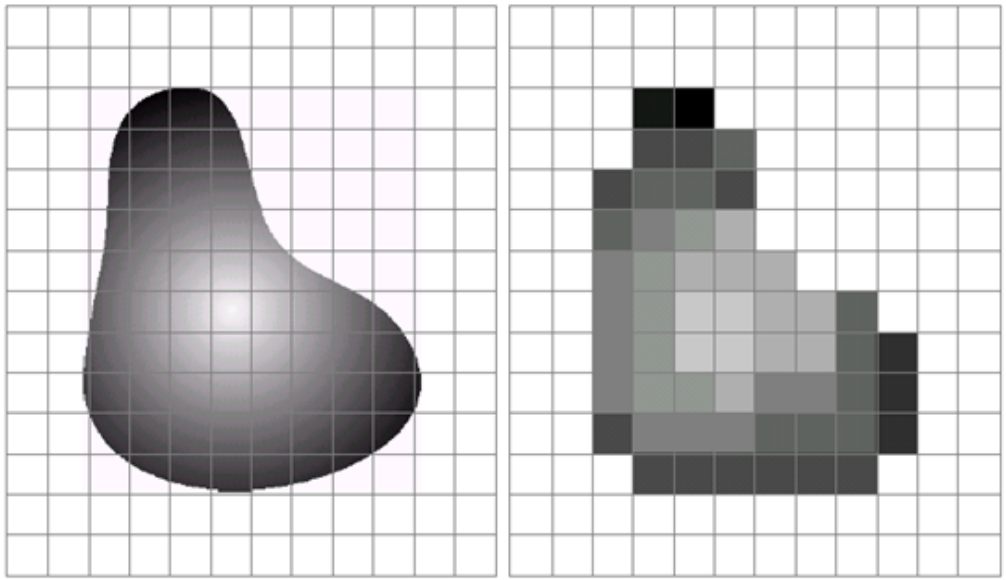


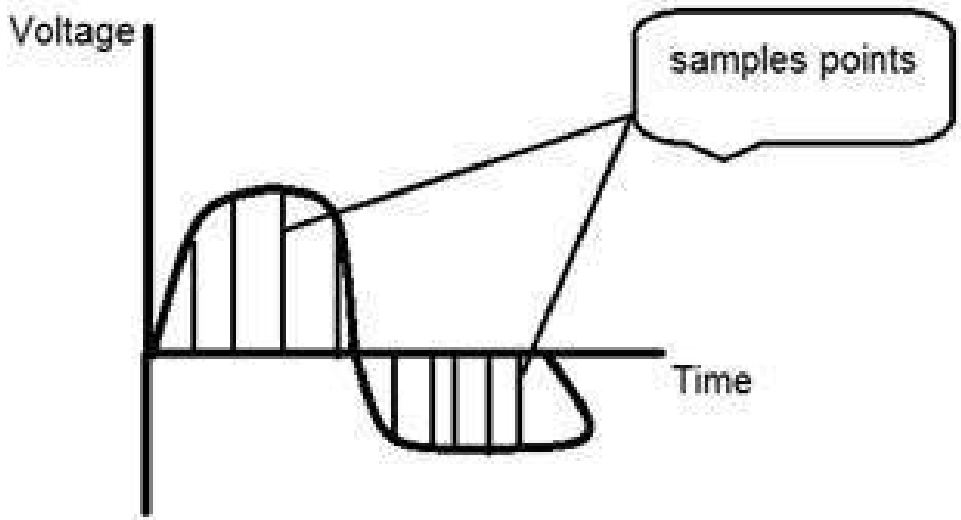
Image Sampling And Quantisation

Remember that a digital image is always only an approximation of a real world scene



Sampling

- The term refers to take samples
- We digitize x axis *i.e.*, independent variable
- In case of equation $y = sin(x)$, it is done on x variable



- Up sampling and down sampling
- How much samples are needed?
- effect of noise?

Pixels and Zooming

- Smallest element of an image.
- Pixel can store a value proportional to the light intensity at that particular location.
- Pixels = total no of rows $*$ total no of columns.
- increase quantity of pixels, so that when you zoom an image , you will see more detail.
- Oversampling
- difference between zooming and sampling?

Resolution

Pixel Resolution

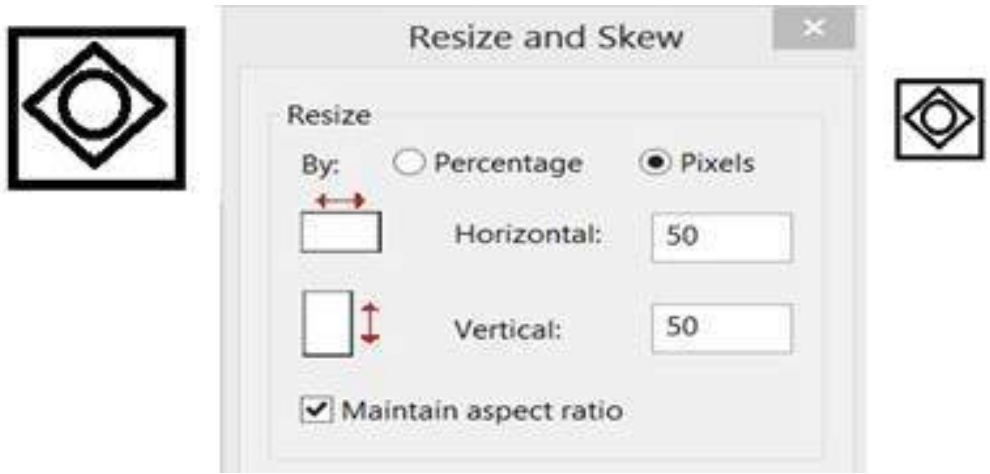
- Pixel resolution , spatial resolution , temporal resolution , spectral resolution
- Monitor resolution of 800 x 600 , 640 x 480 *etc.*
- Pixel resolution: total number of count of pixels in an digital image. *e.g.*, M X N.
- Two numbers; width of the picture, or the pixels across columns. Second number is height of picture, or the pixels across its width.

MegaPixel

- Column pixels (width) X row pixels (height) / 1 Million.
- Size = pixel resolution X bpp (bits per pixel)
- For an image of dimension: 2500 X 3192.
- Its pixel resolution = 2500 * 3192 = 7982350 bytes.
- Dividing it by 1 million = 7.9 = 8 mega pixel (approximately).

Aspect Ratio

- Ratio between width of an image and the height of an image. *e.g.*, 1.33:1, 2.00:1, *etc.*
- Advantages?



- You are given an image with aspect ratio of 6:2 of an image of pixel resolution of 480000 pixels given the image is an gray scale image.
  - Resolve pixel resolution to calculate the dimensions of image
  - Calculate the size of the image

Solution

- Aspect ratio: c:r = 6:2
- Pixel resolution: c * r = 480000
- Bits per pixel: grayscale image = 8bpp
- Number of rows, cols = ?

Equation 1. c:r = 6:2 → c = 6 r / 2

Equation 2. c = 480000/r

Comparing both equations → $\frac{6r}{2} = \frac{480000}{r}$

$$r^2 = \sqrt{\frac{480000*2}{6}}$$

That gives r = 400.

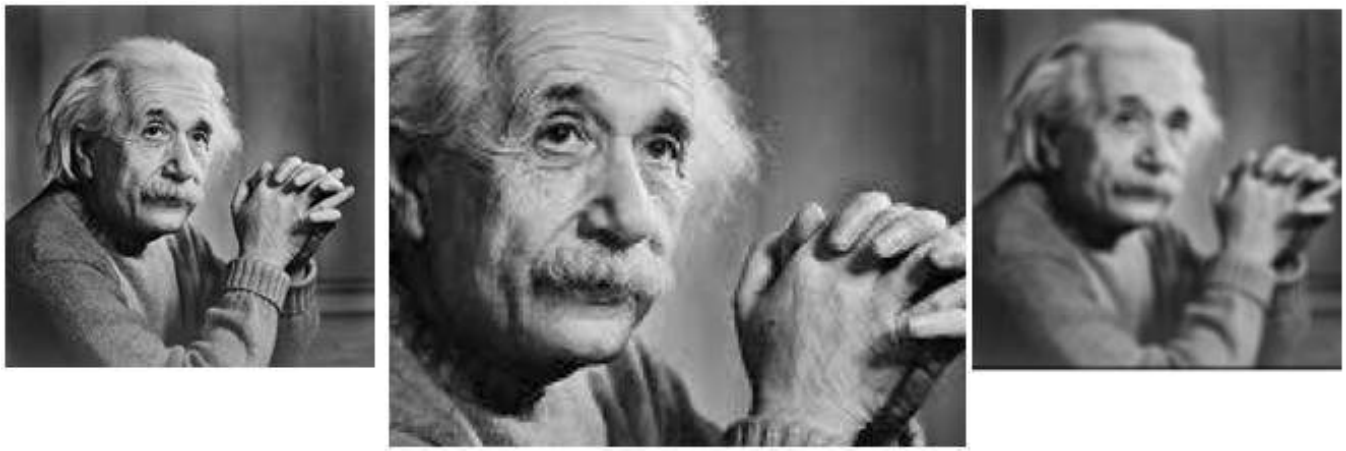
Put r in equation 1, we get → c = 1200.

So rows = 400 cols = 1200.

- Size = rows * cols * bpp = 400 * 1200 * 8 = 3840000 bits
- Size of image in bytes = 480000 bytes; 48 kb (approx).

Spatial Resolution

- Clarity of an image cannot be determined by the pixel resolution. The number of pixels in an image does not matter.
- Definition: Smallest discernible detail in an image. *i.e.*, number of independent pixels values per inch.



Spatial Resolution

- The spatial resolution of an image is determined by how sampling was carried out
- Spatial resolution simply refers to the smallest discernible detail in an image
  - Vision specialists will often talk about pixel size
  - Graphic designers will talk about dots per inch (DPI)
  - Dots per inch or DPI is usually used in monitors.
  - Lines per inch or LPI is usually used in laser printers.
  - Pixel per inch or PPI is measure for different devices such as tablets , Mobile phones *etc.*

Pixels per inch

- Samsung galaxy s4 has PPI or pixel density of 441
  Diagonal resolution:

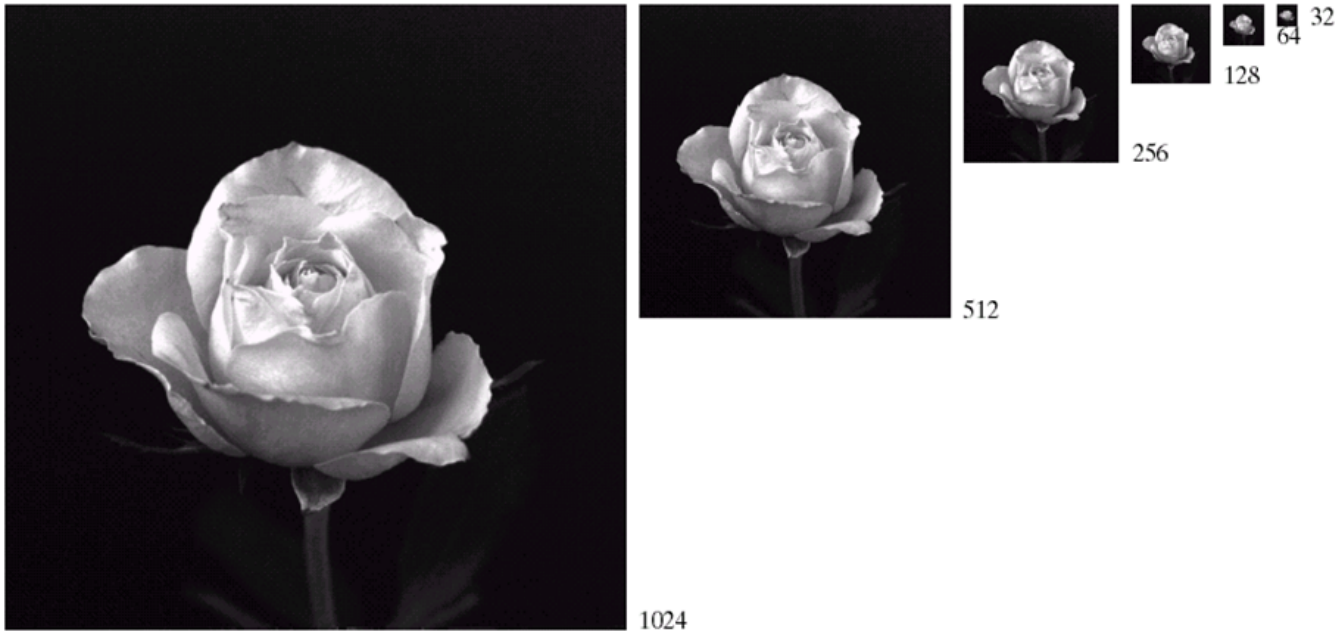
$$c = \sqrt{a^2 + b^2} \tag{1}$$

- Where a and b are height and width resolutions in pixel: 1080 x 1920 pixels: C = 2202.90717;
- PPI = c / diagonal size in inches
- The diagonal size in inches of Samsun galaxy s4 is 5.0 inches.
- PPI = 2202.90717/5.0= PPI = 440.58 = PPI = 441 (approx)

Dots per inch

- DPI or dots per inch is a measure of spatial resolution of printers.
- DPI means that how many dots of ink are printed per inch when an image get printed out from the printer.
- Remember , it is not necessary that each Pixel per inch is printed by one dot per inch. There may be many dots per inch used for printing one pixel. The reason behind this that most of the color printers uses CMYK model. The colors are limited. Printer has to choose from these colors to make the color of the pixel whereas within pc , you have hundreds of thousands of colors.
- The higher is the dpi of the printer , the higher is the quality of the printed document or image on paper.
- Usually some of the laser printers have dpi of 300 and some have 600 or more.

Spatial Resolution



Spatial Resolution

Spatial Resolution: 1024

Spatial Resolution

Spatial Resolution: 512

Spatial Resolution

Spatial Resolution: 256

Spatial Resolution

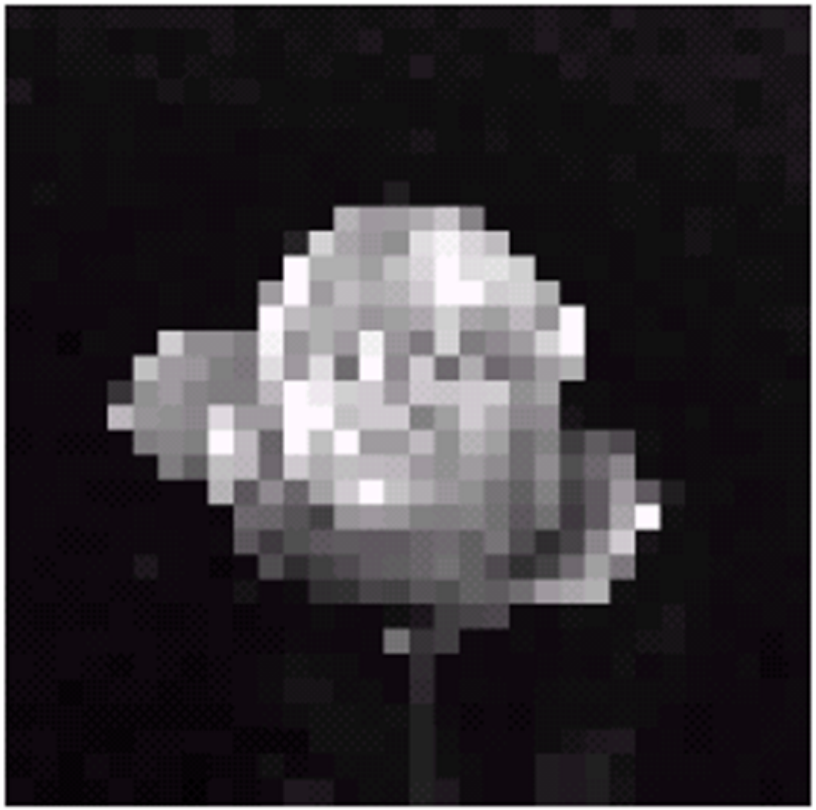
Spatial Resolution: 128

Spatial Resolution

Spatial Resolution: 64

Spatial Resolution

Spatial Resolution: 32

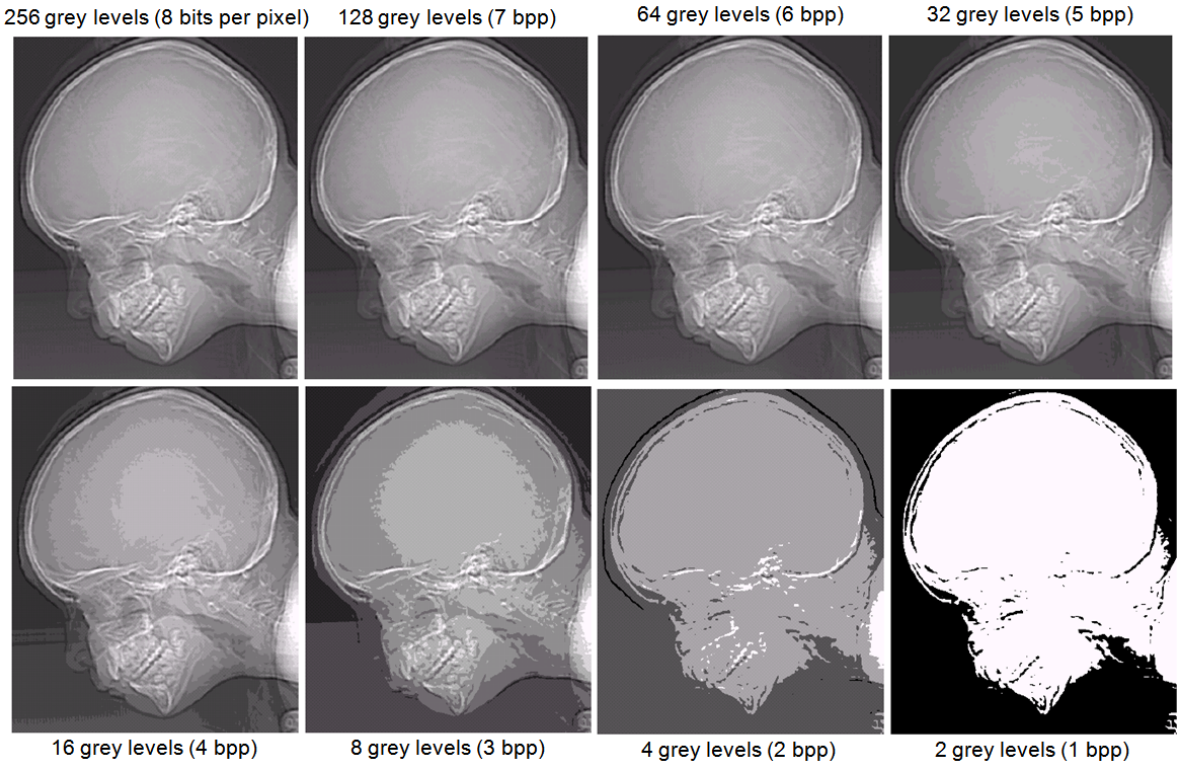


Intensity Level Resolution

- Intensity level resolution refers to the number of intensity levels used to represent the image
  - The more intensity levels used, the finer the level of detail discernable in an image
  - Intensity level resolution is usually given in terms of the number of bits used to store each intensity level

Bits	Intensity Levels	Examples
1	2	(0, 1)
2	4	(00, 01, 10, 11)
4	16	(0000, 0101, 1111)
8	256	(00110011, 01010101)
16	65536	(1010101010101010)

Intensity Level Resolution



Resolution: How Much Is Enough?

- The big question with resolution is always how much is enough?
  - This all depends on what is in the image and what you would like to do with it
  - Key questions include
    - Does the image look aesthetically pleasing?
    - Can you see what you need to see within the image?

Resolution: How Much Is Enough?

Figure: The picture on the right is fine for counting the number of cars, but not for reading the number plate

Resolution: How Much Is Enough?

Figure: The picture on the right is fine for counting the number of cars, but not for reading the number plate

Resolution: How Much Is Enough?

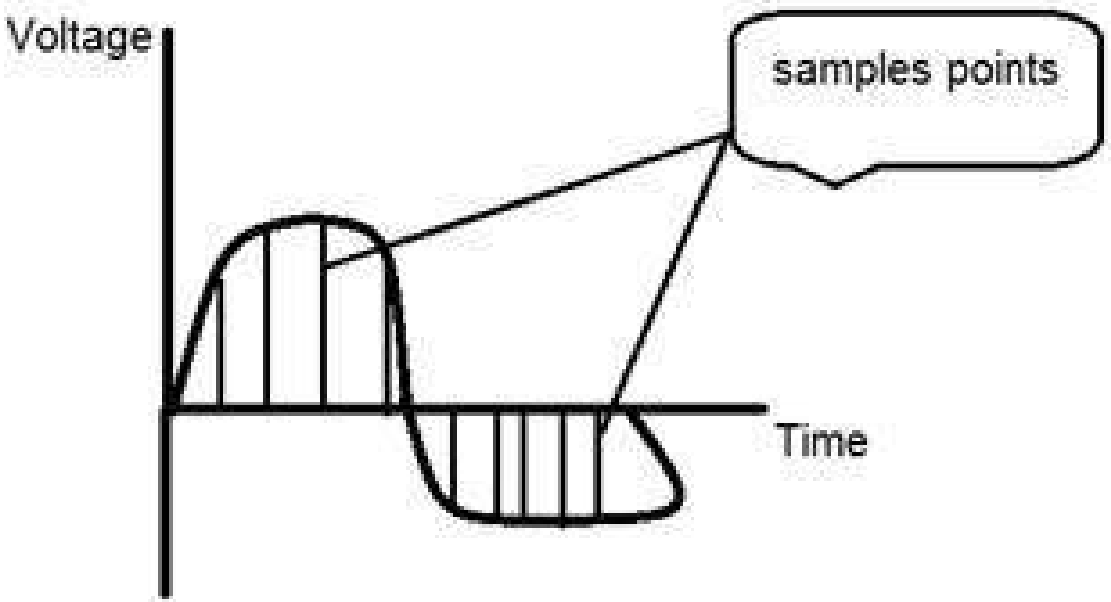
Low Detail

Medium Detail

High Detail

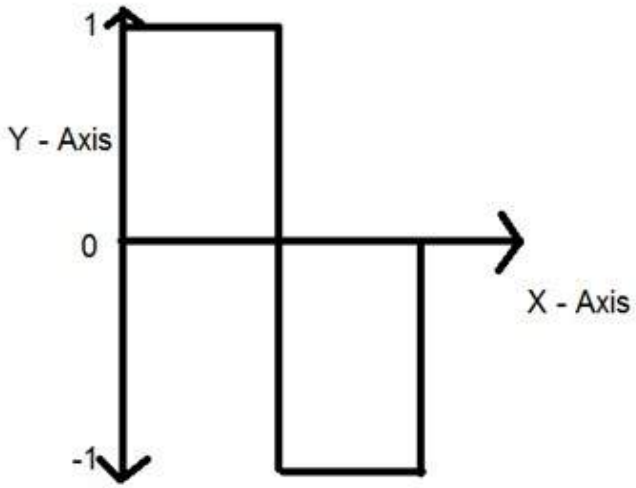
Quantization

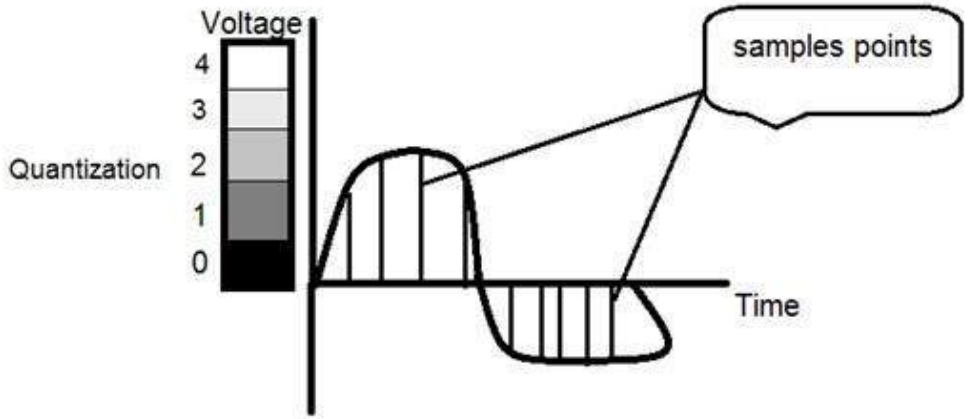
- Gray level resolution is found on the y axis of the signal.
- Sampling is done on x axis. And quantization is done in Y axis. So that means digitizing the gray level resolution of an image is done in quantization.



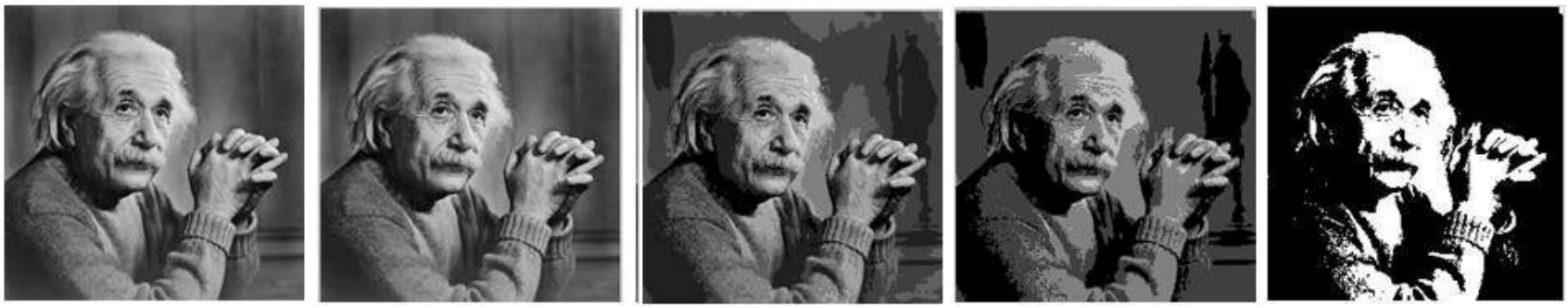
Quantization

Dividing a signal into quanta(partitions).





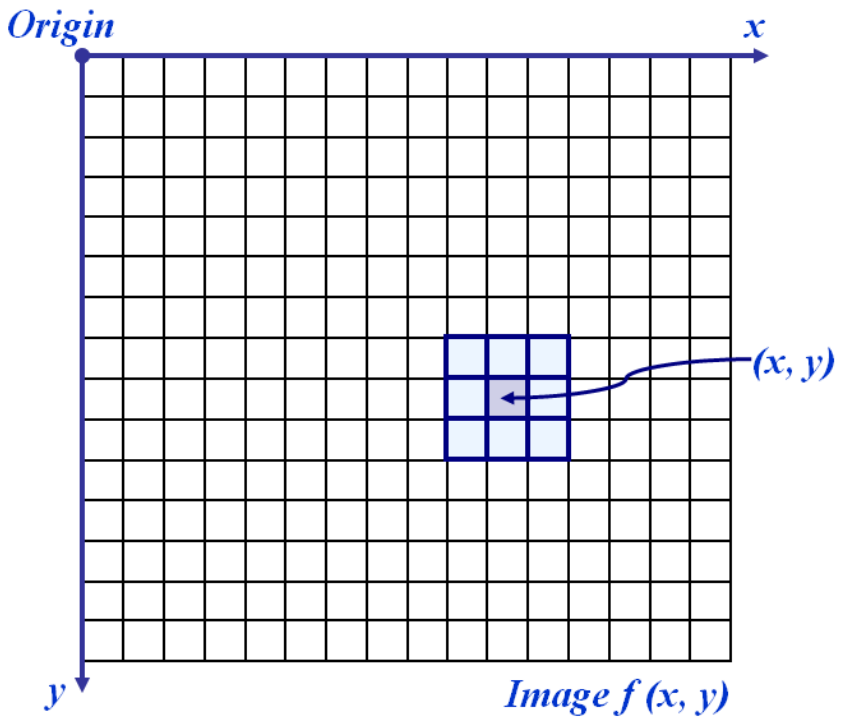
Quantization



Outline

Basic Spatial Domain Image Enhancement

- Most spatial domain enhancement operations can be reduced to the form $g(x, y) = T[f(x, y)]$ where $f(x, y)$ is the input image, $g(x, y)$ is the processed image and T is some operator defined over some neighbourhood of (x, y)



Point Processing

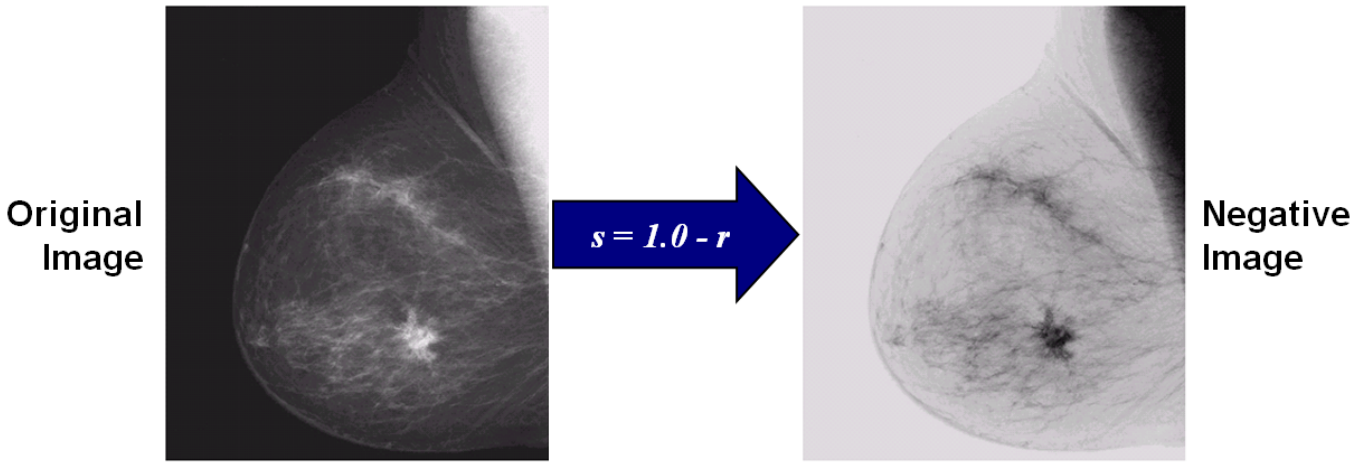
- The simplest spatial domain operations occur when the neighbourhood is simply the pixel itself
- In this case T is referred to as a grey level transformation function or a point processing operation
  - Point processing operations take the form

$$s = T(r) \tag{2}$$

where s refers to the processed image pixel value and r refers to the original image pixel value

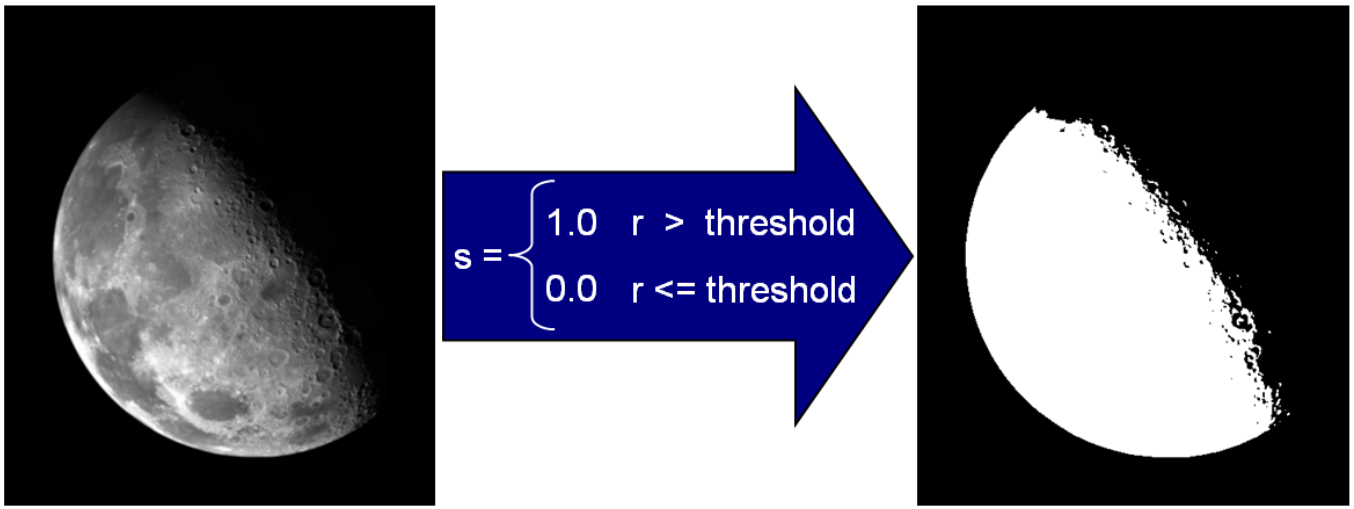
Point Processing Example: Negative Image

- Negative images are useful for enhancing white or grey detail embedded in dark regions of an image
  - Note how much clearer the tissue is in the negative image of the mammogram below



Point Processing Example: Thresholding

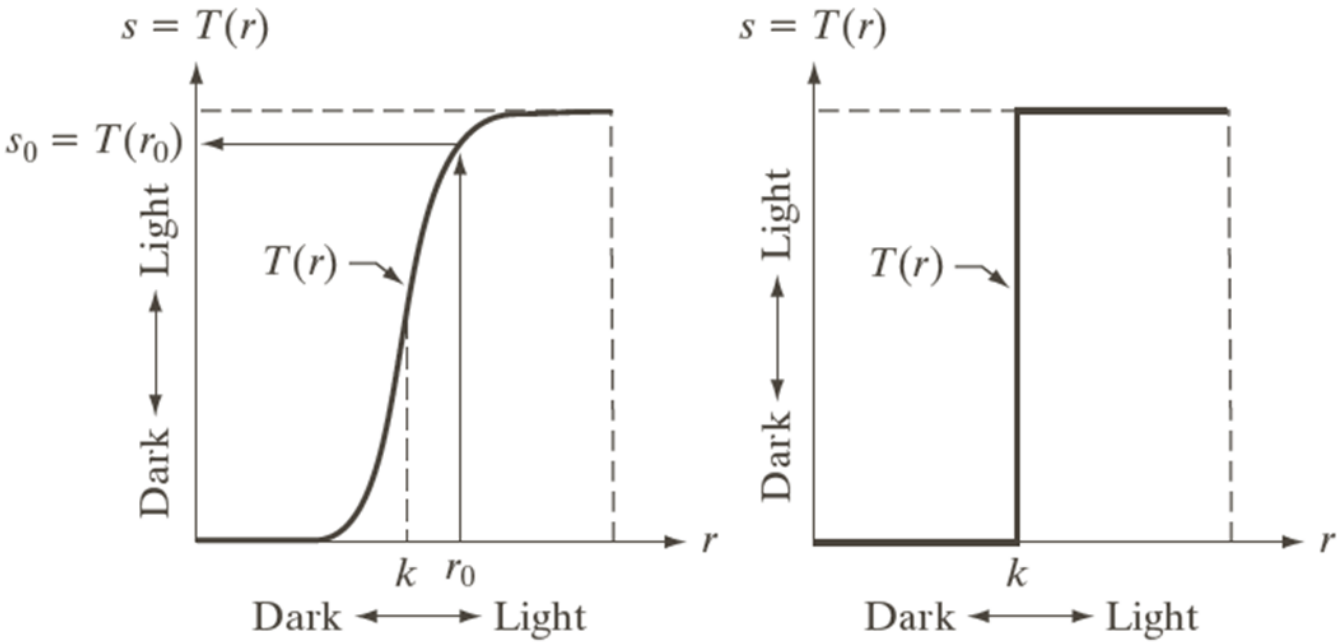
- Thresholding transformations are particularly useful for segmentation in which we want to isolate an object of interest from a background



Outline

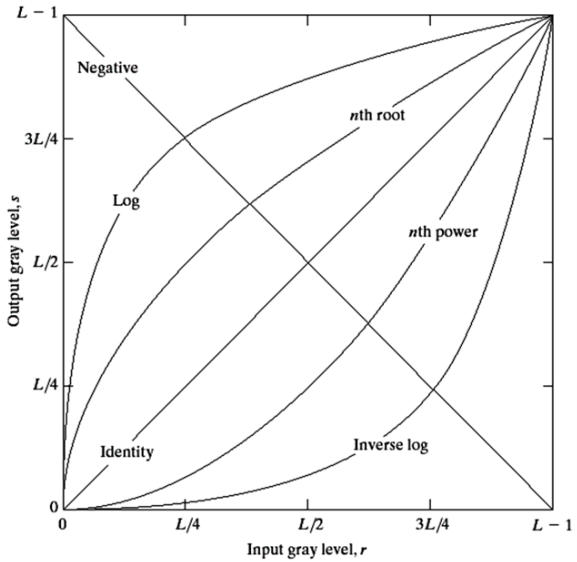
Intensity Transformations

of interest from a background



Basic Grey Level Transformations

- There are many different kinds of grey level transformations
- Three of the most common are shown here
  1. Linear
     - Negative/Identity
  2. Logarithmic
     - Log/Inverse log
  3. Power law
     - *nth* power/ *nth* root



Logarithmic Transformations

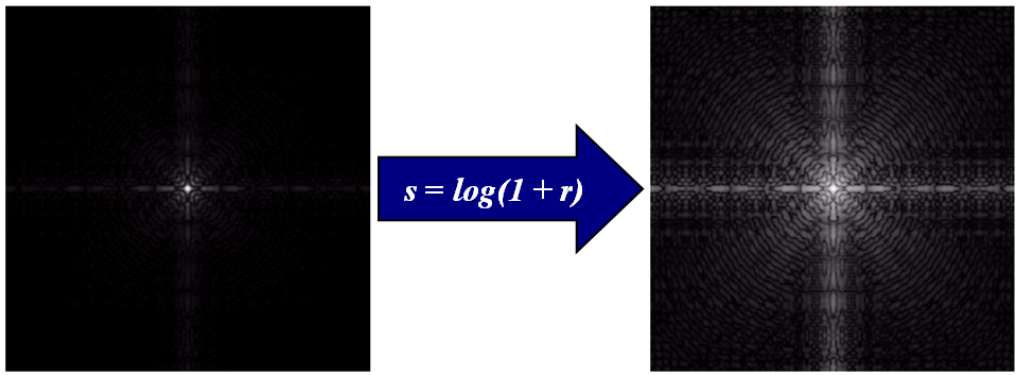
- The general form of the log transformation is

$$s = c * log(1 + r) \tag{3}$$

- The log transformation maps a narrow range of low input grey level values into a wider range of output values
- The inverse log transformation performs the opposite transformation

Logarithmic Transformations

- Log functions are particularly useful when the input grey level values may have an extremely large range of values
- In the following example the Fourier transform of an image is put through a log transform to reveal more detail

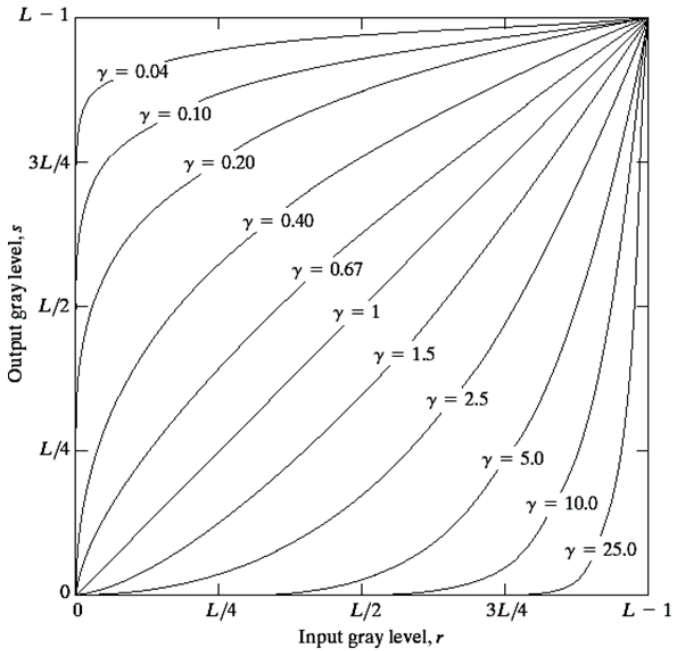


Power Law Transformations

- Power law transformations have the following form

$$s = c * r^{\gamma} \tag{4}$$

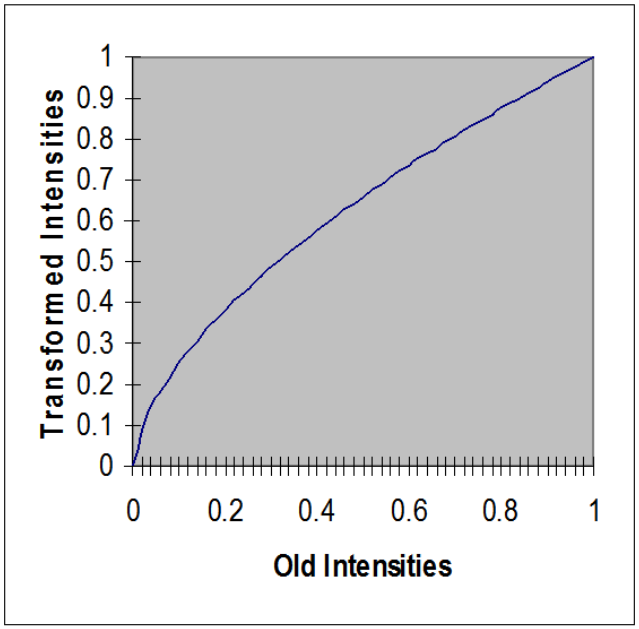
- Map a narrow range of dark input values into a wider range of output values or vice versa
- Varying γ gives a whole family of curves



Power Law Example

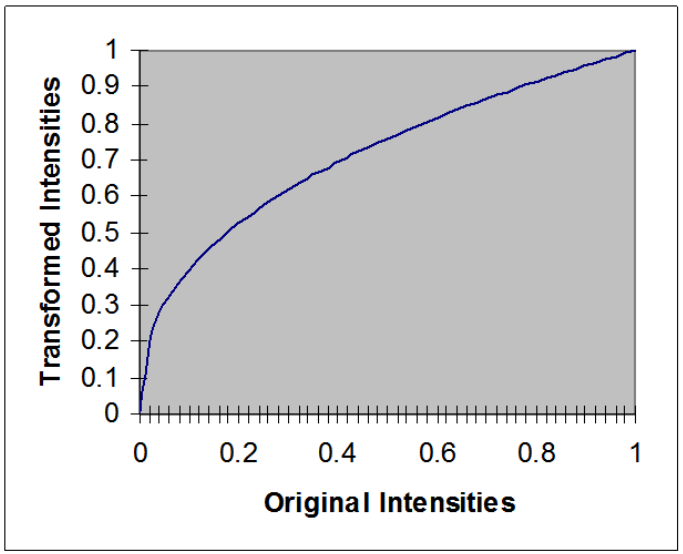
Power Law Example

$$\gamma = 0.6$$



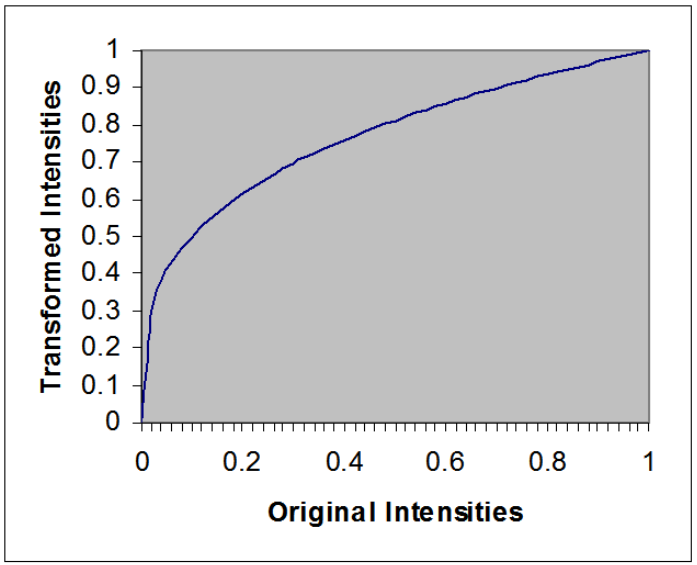
Power Law Example

$$\gamma = 0.4$$



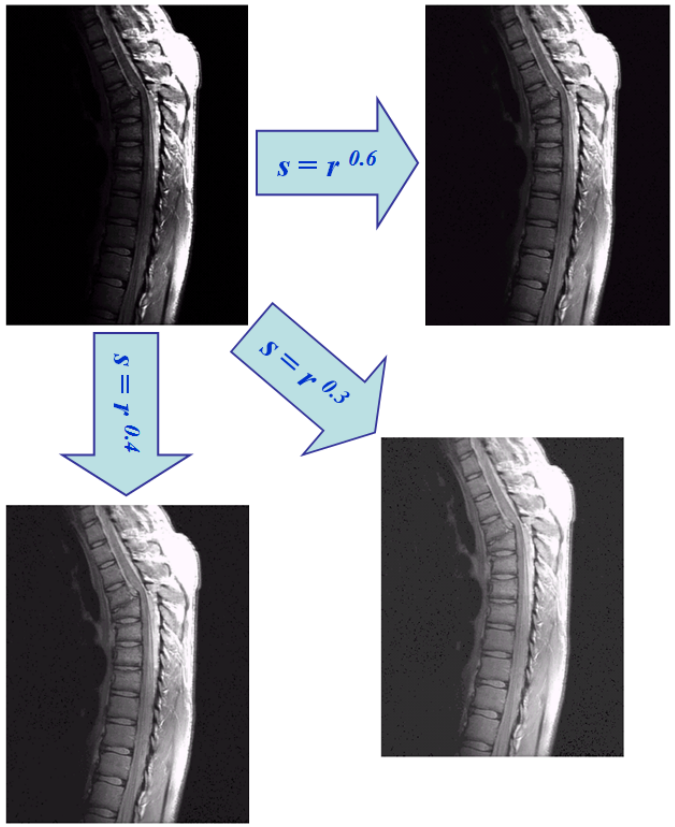
Power Law Example

$$\gamma = 0.3$$



Power Law Example

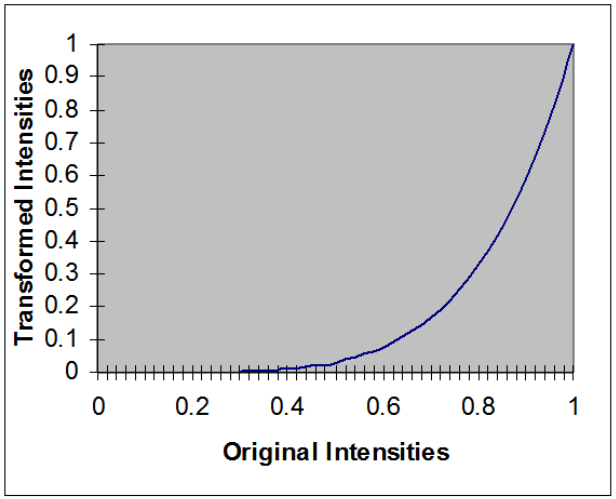
Magnetic Resonance (MR) image of a fractured human spine.
Different curves highlight different details.



Power Law Example

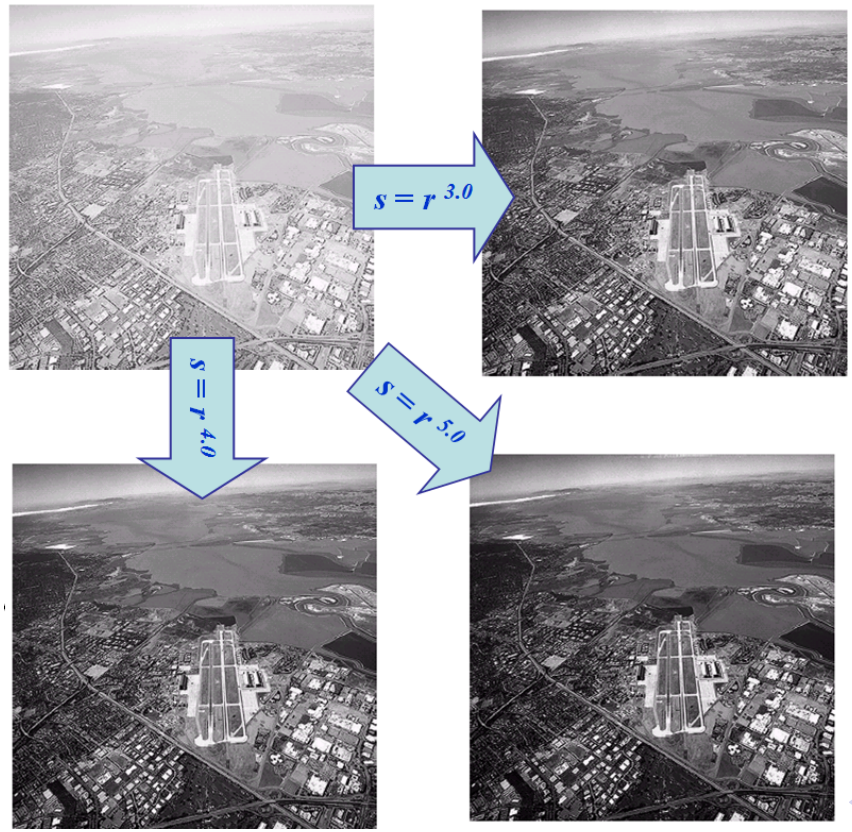
Power Law Example

$$\gamma = 5.0$$



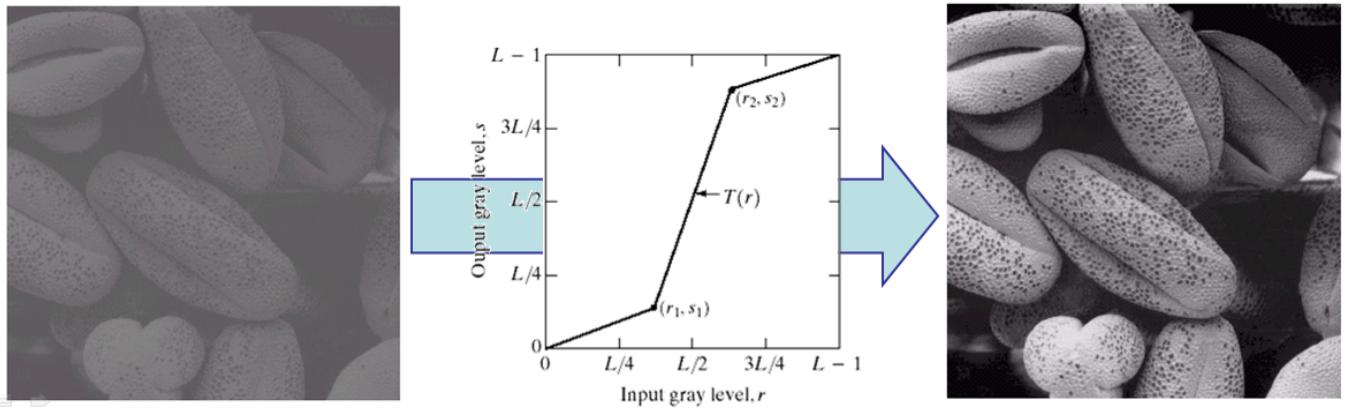
Power Law Example

- An aerial photo of a runway.
- This time power law transforms are used to darken the image.
- Different curves highlight different details.



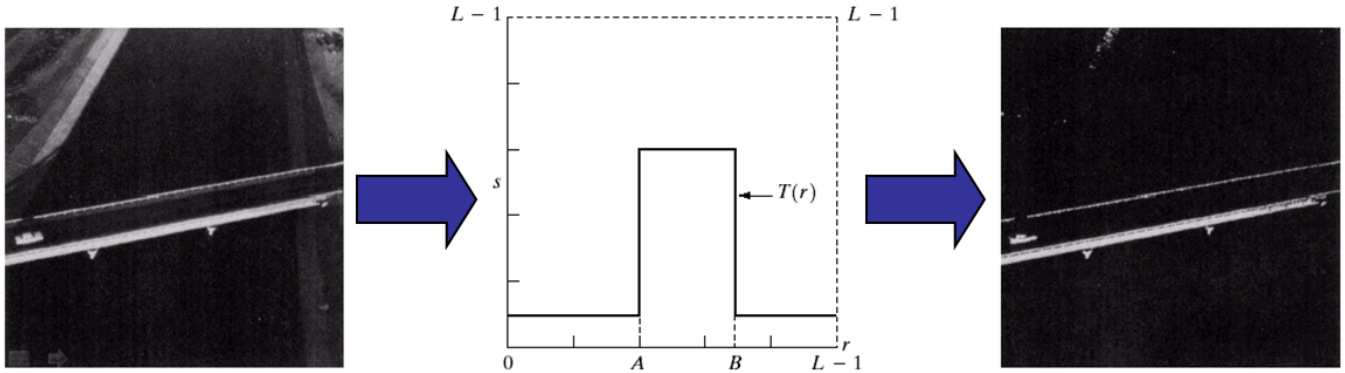
Piecewise Linear Transformation Functions

- Rather than using a well defined mathematical function we can use arbitrary user-defined transforms
- The images below show a contrast stretching linear transform to add contrast to a poor quality image



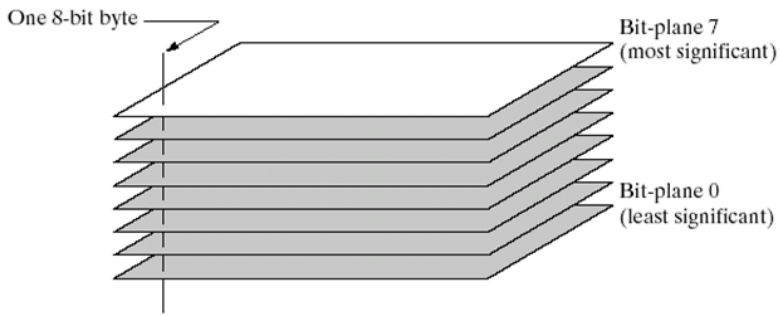
Piecewise Linear Transformation Functions

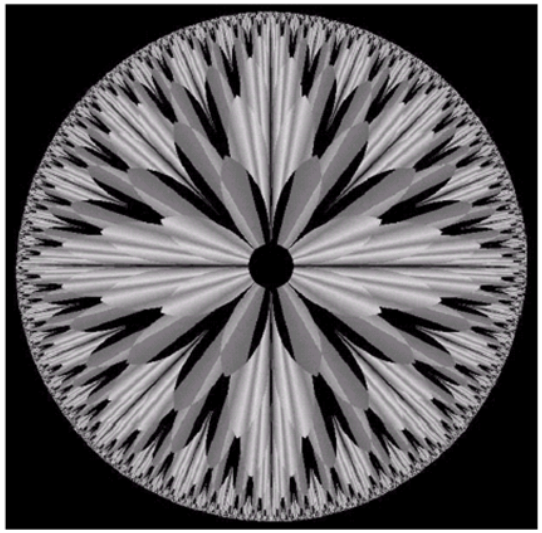
- Highlights a specific range of grey levels
  - Similar to thresholding
  - Other levels can be suppressed or maintained
  - Useful for highlighting features in an image



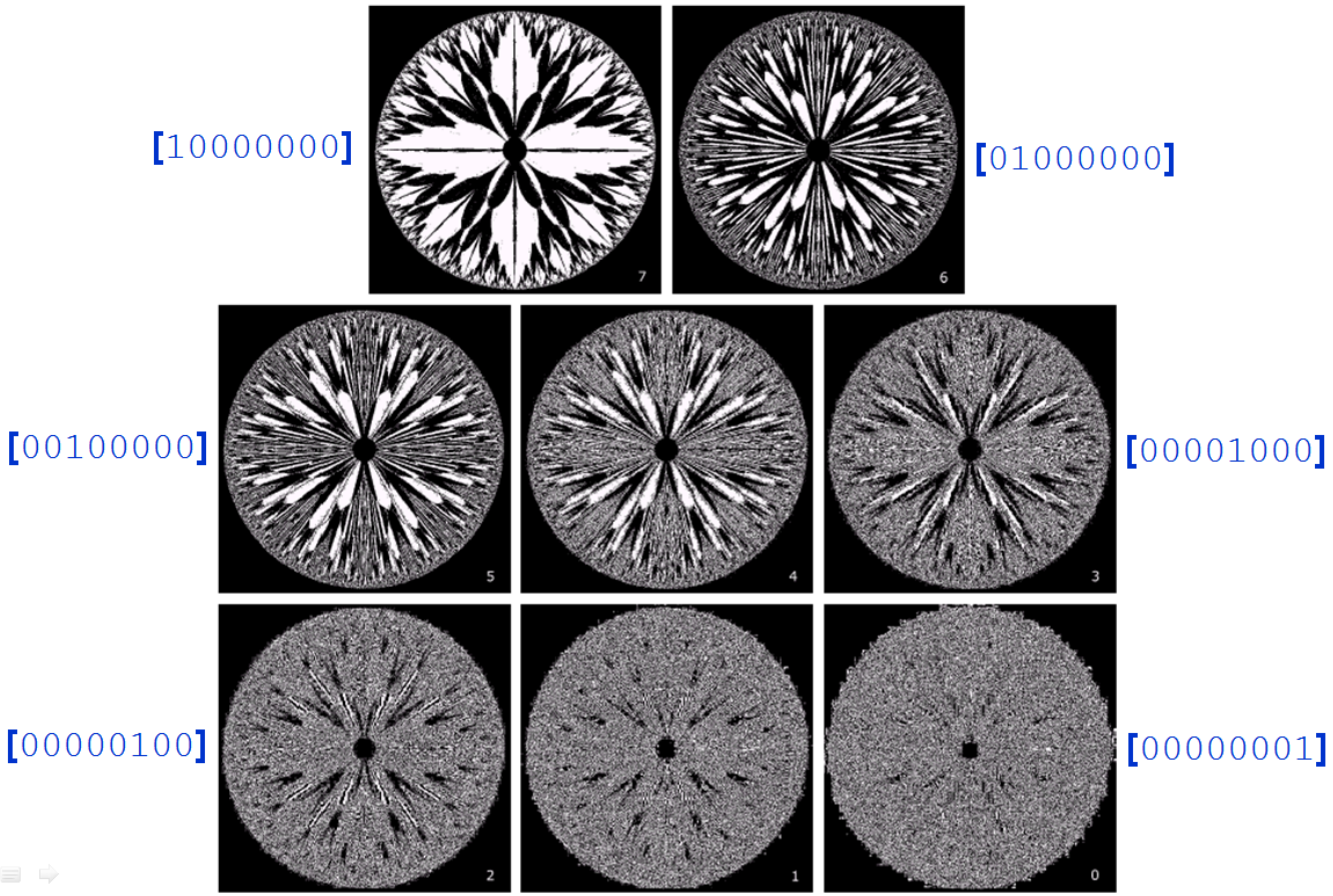
Bit Plane Slicing

- Often by isolating particular bits of the pixel values in an image we can highlight interesting aspects of that image
  - Higher-order bits usually contain most of the significant visual information
  - Lower-order bits contain subtle details





Bit Plane Slicing



Bit Plane Slicing

a b c
d e f
g h i

FIGURE 3.14 (a) An 8-bit gray-scale image of size 500 × 1192 pixels. (b) through (i) Bit planes 1 through 8, with bit plane 1 corresponding to the least significant bit. Each bit plane is a binary image.

Matlab Desktop

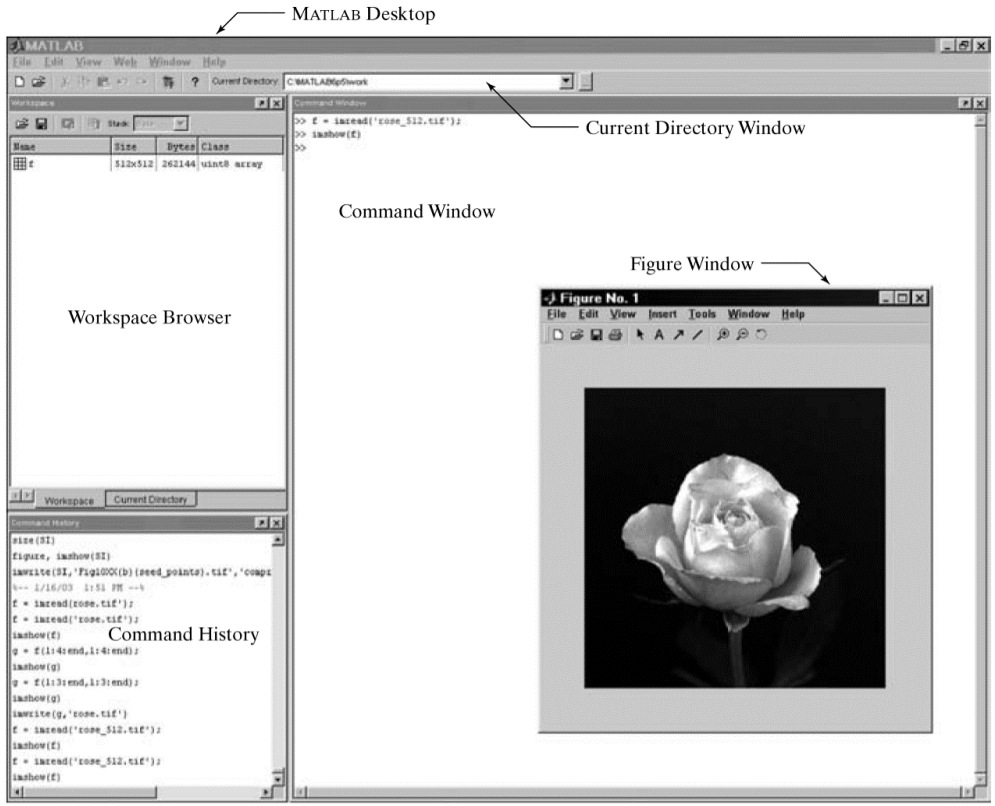
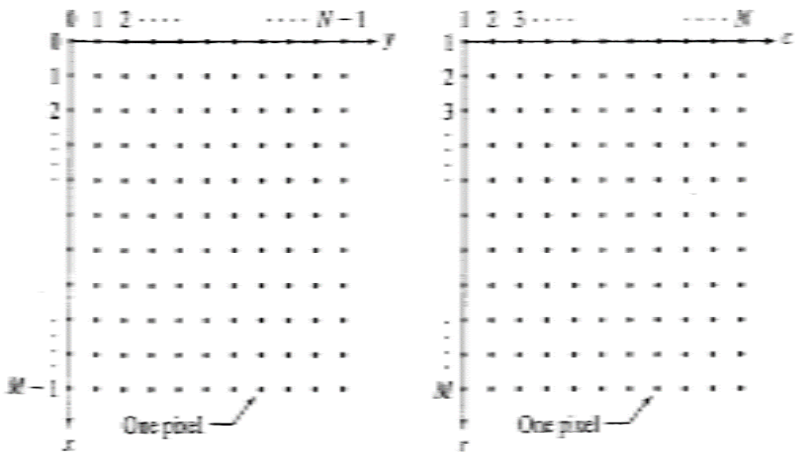


FIGURE 1.1 The MATLAB desktop and its principal components.

Image Representation



a b

FIGURE 2.1 Coordinate conventions used (a) in many image processing books, and (b) in the Image Processing Toolbox.

Image Processing in MATLAB

- Images can be conveniently represented as matrices in MATLAB.
- Open an image as a matrix using ?imread? command.
- The matrix may simply be m x n form.
- The image processing may be done simply by matrix calculation or matrix manipulation.
- Image may be displayed with ?imshow? command.
- Changes in an image may then be saved with ?imwrite? command.

The Image Processing Toolbox (IPT).

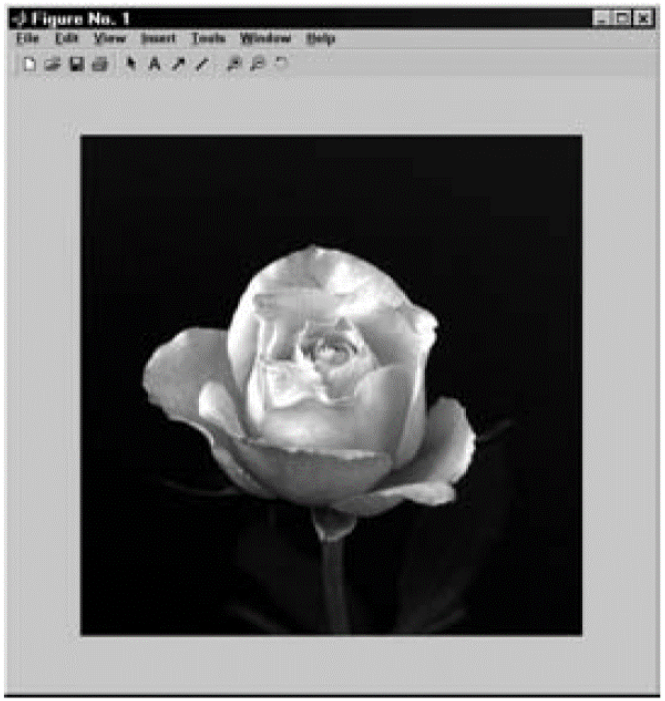
- IPT is a collection of functions that extend the capability of the MATLAB numeric computing environment.
- The toolbox supports a wide range of image processing operations, including Spatial image transformations, Morphological operations, Neighborhood and block operations, Linear filtering and filter design, Transforms, Image analysis and enhancement, Image registration, Deblurring Region of interest operations, etc.

Images as Matrices

- Defining a matrix:
  » A = [1 2 3; 3 4 5; 6 7 8] % Row-major form.
- Perform many standard operations on the matrix. e.g, you can find the inverse of a matrix :
  » inv(A)
- You can find an approximation to the eigen values of a matrix:
  » eig(A)
  » [v,e] = eig(A)
  » diag(e)

Displaying Image

- f = imread('rose.tif');
  imshow(f);



Displaying Image Information

- f = imread("Image_Name.extension");
- imshow(f)
- impixelinfo
- size(f): ans = 1024 1024

M,N = size(f);

- whos f
- Name Size Bytes Class Attributes
  r 1024*x*1024 1048576 *uint*8

Displaying Image Information

imfinfo (bubbles.jpg)

ans =

Filename:	'bubble.jpg'
FileModDate:	'14-Jan-2008 17:08:08'
FileSize:	7904
Format:	'jpg'
FormatVersion:	''
Width:	720
Height:	688
BitDepth:	8
ColorType:	'grayscale'
FormatSignature:	''
NumberOfSamples:	1
CodingMethod:	'Huffman'
CodingProcess:	'Sequential'
Comment:	{}

Try this

```
>> K imfinfo('bubbles.jpg');
>> K.FileSize

ans =

      7904
```

Writing Images

Syntax:

variable=imread("Image_name.extension");

imwrite(variable,"Name.ext");

Step1:Read

» b = imread('bubbles.tif');

Step2:Write

» imwrite(b,'bubbles.png');

» imwrite(b,'bubbles.jpg');

» imwrite(b,'bubbles', 'jpg');

Image Types in Matlab

- Three types of images i.e. black & white, grey scale and colored.
- Black & White images are called binary images, containing 1 for white and 0 for black.
- Grey scale images are called intensity images, containing numbers in the range of 0 to 255 or 0 to 1.
- Colored images may be represented as RGB Image or Indexed Image.
- In RGB Images there exist three indexed images.

Image Types in Matlab Con...

- First image contains all the red portion of the image, second green and third contains the blue portion.
- So for a 640 x 480 sized image the matrix will be 640 x 480 x 3.
- It actually exist of two matrices namely image matrix and map matrix.
- Each color in the image is given an index number and in image matrix each color is represented as an index number.
- Map matrix contains the database of which index number belongs to which color.

Image Formats

Format Name	Description	Recognized Extensions
TIFF	Tagged Image File Format	`.tif`, `.tiff`
JPEG	Joint Photographic Experts Group	`.jpg`, `.jpeg`
GIF	Graphics Interchange Format†	`.gif`
BMP	Windows Bitmap	`.bmp`
PNG	Portable Network Graphics	`.png`
XWD	X Window Dump	`.xwd`

† `GIF` is supported by `imread`, but not by `imwrite`.

Data Classes

Name	Description
`double`	Double-precision, floating-point numbers in the approximate range -10^{308} to 10^{308} (8 bytes per element).
`uint8`	Unsigned 8-bit integers in the range $[0, 255]$ (1 byte per element).
`uint16`	Unsigned 16-bit integers in the range $[0, 65535]$ (2 bytes per element).
`uint32`	Unsigned 32-bit integers in the range $[0, 4294967295]$ (4 bytes per element).
`int8`	Signed 8-bit integers in the range $[-128, 127]$ (1 byte per element).
`int16`	Signed 16-bit integers in the range $[-32768, 32767]$ (2 bytes per element).
`int32`	Signed 32-bit integers in the range $[-2147483648, 2147483647]$ (4 bytes per element).
`single`	Single-precision floating-point numbers with values in the approximate range -10^{38} to 10^{38} (4 bytes per element).
`char`	Characters (2 bytes per element).

Converting between Image Classes & Types

Name	Converts Input to:	Valid Input Image Data Classes
`im2uint8`	`uint8`	`logical`, `uint8`, `uint16`, and `double`
`im2uint16`	`uint16`	`logical`, `uint8`, `uint16`, and `double`
`mat2gray`	`double` (in range $[0, 1]$)	`double`
`im2double`	`double`	`logical`, `uint8`, `uint16`, and `double`
`im2bw`	`logical`	`uint8`, `uint16`, and `double`

Converting between Image Classes & Types

Example: Consider the following 2 X 2 image f of class double:

Try This:

```
>> f  = [-0.5  0.5; 0.75  1.5]
f =
        -0.5        0.5
        0.75        1.5

>> g = im2uint8(f)

g =
           0        128
         191        255
```

```
>> h = uint8([25  50; 128  200]);
>> g = im2double(h);
g =
      0.0980      0.1961
      0.4706      0.7843
```

Array Indexing

Example:

```
>> v = [1  3  5  7  9]
v =
          1  3  5  7  9
>> v(2)   "2nd  index"
ans =
           3
>> v(2:4) "2nd  to 4th  index"
ans =
          3  5  7
```

Guess the output of: v(3:end) v(1:2:end) v(end:-2:1)

Matrix Indexing

Example:

```
>> A = [1  2  3; 4  5  6; 7  8  9]
A =
        1  2  3
        4  5  6
        7  8  9

>> A(2, 3)
ans =
        6

>> R2 = A(2, :)
R2 =
        4  5  6
```

Guess the output of: `A(1:2, 1:3)` `A(end, end-2)` `A(:, 3)`

Try this:

```
C3 = A(:, 3)
C3 =
        3
        6
        9
```

Standard Arrays

```
>> A = 5*ones(3, 3)
A =
     5     5     5
     5     5     5
     5     5     5
```

```
>> magic(3)
ans =
     8     1     6
     3     5     7
     4     9     2
```

```
>> B = rand(2, 4)
B =
    0.2311    0.4860    0.7621    0.0185
    0.6068    0.8913    0.4565    0.8214
```

Array and Matrix Arithmatic Operators

Operator	Name	MATLAB Function	Comments and Examples
+	Array and matrix addition	`plus(A, B)`	a + b, A + B, or a + A.
–	Array and matrix subtraction	`minus(A, B)`	a – b, A – B, A – a, or a – A.
.*	Array multiplication	`times(A, B)`	`C = A.*B, C(I, J) = A(I, J)*B(I, J).`
*	Matrix multiplication	`mtimes(A, B)`	`A*B`, standard matrix multiplication, or `a*A`, multiplication of a scalar times all elements of A.
./	Array right division	`rdivide(A, B)`	`C = A./B, C(I, J) = A(I, J)/B(I, J).`
.\	Array left division	`ldivide(A, B)`	`C = A.\B, C(I, J) = B(I, J)/A(I, J).`
/	Matrix right division	`mrdivide(A, B)`	`A/B` is roughly the same as `A*inv(B)`, depending on computational accuracy.
\	Matrix left division	`mldivide(A, B)`	`A\B` is roughly the same as `inv(A)*B`, depending on computational accuracy.
.^	Array power	`power(A, B)`	If `C = A.^B`, then `C(I, J) = A(I, J)^B(I, J).`
^	Matrix power	`mpower(A, B)`	See online help for a discussion of this operator.
.'	Vector and matrix transpose	`transpose(A)`	`A.'`. Standard vector and matrix transpose.
'	Vector and matrix complex conjugate transpose	`ctranspose(A)`	`A'`. Standard vector and matrix conjugate transpose. When A is real `A.' = A'`.
+	Unary plus	`uplus (A)`	+A is the same as 0 + A.
–	Unary minus	`uminus (A)`	–A is the same as 0 – A or –1*A.
:	Colon		Discussed in Section 2.8.

TABLE 2.4
Array and matrix arithmetic operators. Computations involving these operators can be implemented using the operators themselves, as in A + B, or using the MATLAB functions shown, as in `plus (A, B)`. The examples shown for arrays use matrices to simplify the notation, but they are easily extendable to higher dimensions.

Operators

Example:

```
f =
     1     2
     3     4

>> v = f(:) "All column in to one column"
v =
     1
     3
     2
     4
```

Arithmatic Operators

Function	Description
`imadd`	Adds two images; or adds a constant to an image.
`imsubtract`	Subtracts two images; or subtracts a constant from an image.
`immultiply`	Multiplies two images, where the multiplication is between pairs of corresponding image elements; or multiplies a constant times an image.
`imdivide`	Divides two images, where the division is carried out between pairs of corresponding image elements; or divides an image by a constant.
`imabsdiff`	Computes the absolute difference between two images.
`imcomplement`	Complements an image. See Section 3.2.1.
`imlincomb`	Computes a linear combination of two or more images. See Section 5.3.1 for an example.

.def Error: File `figs/operators1.png' not found. \end{frame}
'ators1.png' not found
99.92412pt too high) detected

TABLE 2.5
The image arithmetic functions supported by IPT.

Example:
>> imshow(imcomplement(f))

Relational Operators

Operator	Name
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

TABLE 2.6
Relational operators.

Example:

```
>> A = [1 2; 3 4]
A =
   1   2
   3   4

>> B = [0 2; 3 5;]
B =
   0   2
   3   5

>> A>=B
ans =
   1   1
   1   0
```

Logical Operators

Operator	Name
&	AND
|	OR
~	NOT

TABLE 2.7
Logical operators.

Function	Comments
`xor` (exclusive OR)	The `xor` function returns a 1 only if both operands are logically different; otherwise `xor` returns a 0.
`all`	The `all` function returns a 1 if all the elements in a vector are nonzero; otherwise `all` returns a 0. This function operates columnwise on matrices.
`any`	The `any` function returns a 1 if any of the elements in a vector is nonzero; otherwise `any` returns a 0. This function operates columnwise on matrices.

TABLE 2.8
Logical functions.

Flow Control Statements

Operator	Name
&	AND
|	OR
~	NOT

TABLE 2.7
Logical operators.

Function	Comments
`xor` (exclusive OR)	The `xor` function returns a 1 only if both operands are logically different; otherwise `xor` returns a 0.
`all`	The `all` function returns a 1 if all the elements in a vector are nonzero; otherwise `all` returns a 0. This function operates columnwise on matrices.
`any`	The `any` function returns a 1 if any of the elements in a vector is nonzero; otherwise `any` returns a 0. This function operates columnwise on matrices.

TABLE 2.8
Logical functions.

Flow Control Statements

Statement	Description
`if`	`if`, together with `else` and `elseif`, executes a group of statements based on a specified logical condition.
`for`	Executes a group of statements a fixed (specified) number of times.
`while`	Executes a group of statements an indefinite number of times, based on a specified logical condition.
`break`	Terminates execution of a `for` or `while` loop.
`continue`	Passes control to the next iteration of a `for` or `while` loop, skipping any remaining statements in the body of the loop.
`switch`	`switch`, together with `case` and `otherwise`, executes different groups of statements, depending on a specified value or string.
`return`	Causes execution to return to the invoking function.
`try...catch`	Changes flow control if an error is detected during execution.

TABLE 2.11 Flow control statements.