CM2208: Scientific Computing
3. Image Processing
3.1. Introduction

Prof. David Marshall

School of Computer Science & Informatics

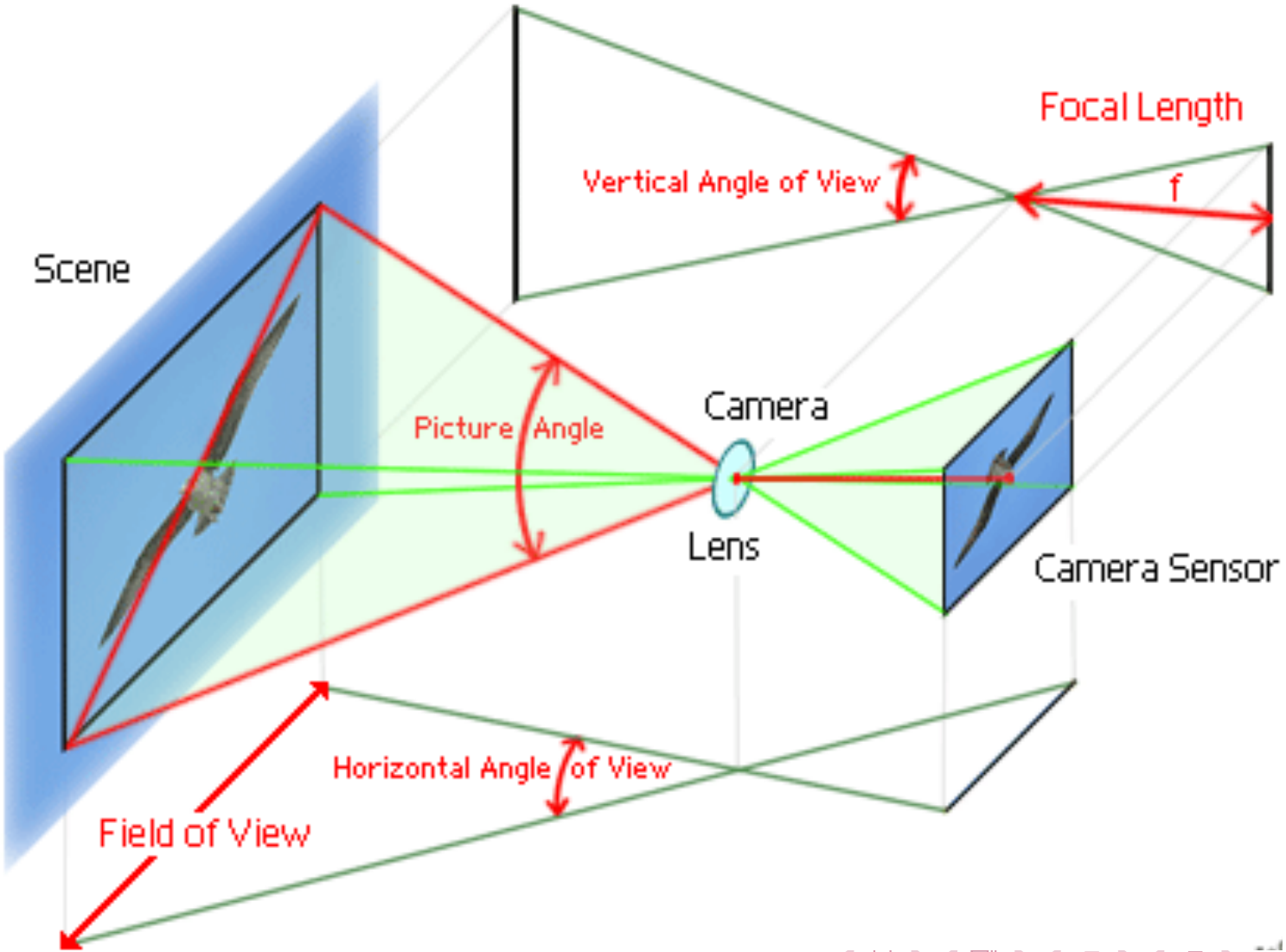
Image Processing

Image Processing Definition

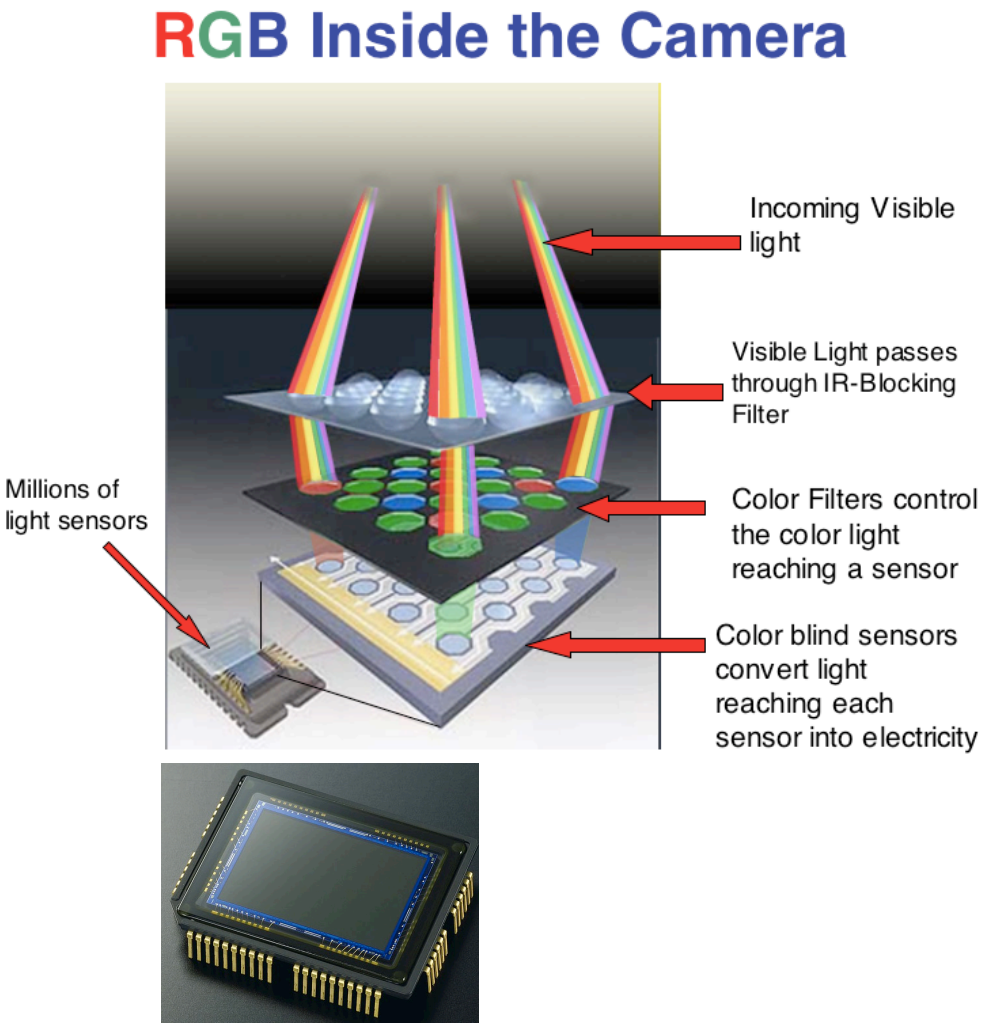
Image processing is any form of **signal processing** for which the input is an **image**, such as a **digital photograph** or **video frame**; the **output** of image processing may be either an **image** or a **set of characteristics** or **parameters** or **features** related to the image.

- Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal processing techniques to it.
- So Image Processing is essentially two-dimensional signal processing— this does involve quite a few new/modified techniques to deal with spatial information in 2D rather 1D data over time.

Imaging: The Camera



Imaging: The Sensor



Basic Image Data Structure

"A picture is worth a thousand words, but it uses up three thousand times the memory."

What is an Image?

- A digital image consists of many picture elements, termed **pixels**.
- The number of pixels determine the quality of the image (**resolution**).
- Higher resolution always yields better quality.

We can regard an **image** as a **two-dimensional light intensity function** $f(x, y)$ where x and y are **spatial** coordinates and the value of f at any point (x, y) is proportional to the colour/grey intensity values or brightness of the image at that point.

Digitised Images

Digitised Images and Pixels

A **digitised image** is one where

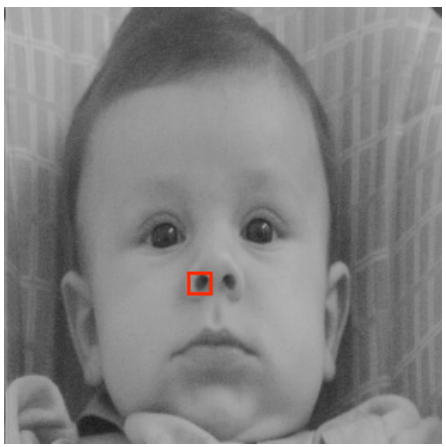
- spatial and colour/greyscale values have been made discrete.
- intensity measured across a regularly spaced grid in x and y directions
- intensities sampled to 8 bits (256 values) per colour/grey scale or index.

For computational purposes, we may think of a digital image as a **two-dimensional array** (**matrix**) where x and y index an image point.

- Each element in the array is called a *Each element in the array is called a* pixel *(picture element).* (picture element).

Images as Matrices

- Images (uncompressed) are represented as a grid of pixels.



99	71	61	51	49	40	35	53	86	99
93	74	53	56	48	46	48	72	85	102
101	69	57	53	54	52	64	82	88	101
107	82	64	63	59	60	81	90	93	100
114	93	76	69	72	85	94	99	95	99
117	108	94	92	97	101	100	108	105	99
116	114	109	106	105	108	108	102	107	110
115	113	109	114	111	111	113	108	111	115
110	113	111	109	106	108	110	115	120	122
103	107	106	108	109	114	120	124	124	132

What do the pixel values represent?

Monochrome/Bit-Map Images

Sample Monochrome Bit-Map Image

- Each pixel is stored as a single bit (0 or 1)
- A 640 x 480 monochrome image requires 37.5 KB of storage.
- *Dithering* is often used for displaying monochrome images

Gray-scale Images

Example of a Gray-scale Bit-map Image

- Each pixel is usually stored as a **byte** (value between **0** to **255**)
- A **dark pixel** will have a **low value**, *e.g.* 10; a **bright one** may be 240, for example.
- A 640 x 480 greyscale image requires around 300 KB of storage.

Dithering (1)

Dithering Process

- **Dithering** is often used when **converting greyscale** images to **monochrome** ones e.g. for **printing**.
- The main strategy is to **replace** a pixel value (from 0 to 255) by a larger pattern (e.g. 4×4) such that the number of printed dots approximates the greyscale level of the original image
- If a pixel is **replaced** by a 4×4 array of dots, the intensities it can approximate from 0 (no dots) to 16 (full dots).
- Given a 4×4 dither matrix e.g.

$$\begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}$$

we can re-map pixel values from 0–255 to a new range 0–16 by dividing the value by (256/17) (and rounding down to integer).

Dithering (2)

Dithering Process Cont.

- A simple approach: replace each pixel by a 4×4 **dots** array (monochrome pixels). If the remapped intensity is $>$ the dither matrix entry, put a dot at the position (set to 1) **otherwise** set to 0.
- Note that the size of the dithered image may be much larger. Since each pixel is replaced by 4×4 array of dots, the image becomes 16 times as large.
- To keep the image size: an ordered dither produces an output pixel with value 1 iff the remapped intensity level just at the pixel position is greater than the corresponding matrix entry.

24-bit (True) Colour Images

Example of 24-Bit Colour Image

- Each pixel is represented by three bytes (e.g., RGB)
- Supports 256 x 256 x 256 possible combined colours (16,777,216)
- A 640 x 480 24-bit colour image would require 921.6 KB of storage
- Some colour images are 32-bit images,
  - the extra byte of data for each pixel is used to store an *alpha* value representing special effect information

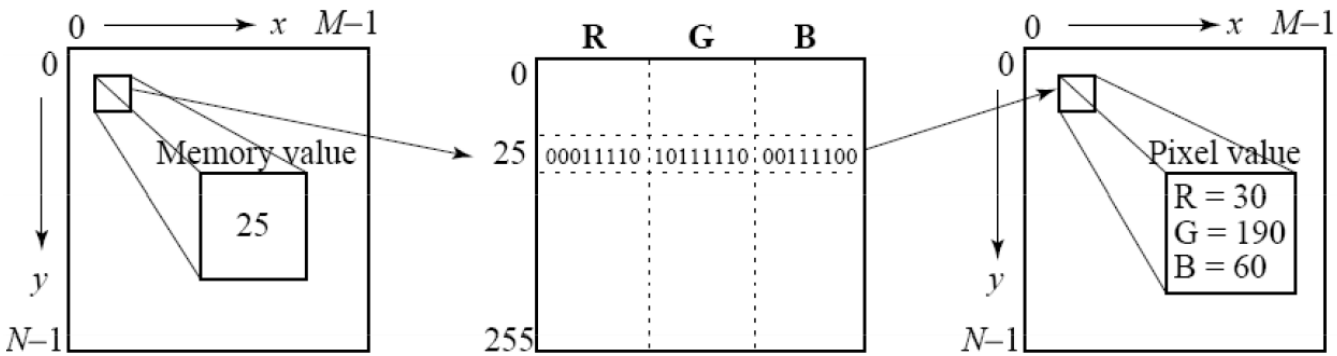
8-bit Colour Images

Example of 8-Bit Colour Image

- One byte for each pixel
- Supports 256 out of the millions possible
- Acceptable colour quality
- Requires Colour Look-Up Tables (LUTs)
- A 640 x 480 8-bit colour image requires 307.2 KB of storage (the same as 8-bit greyscale)

Colour Look-Up Tables (LUTs)

- Store only the index of the colour LUT for each pixel
- Look up the table to find the colour (RGB) for the index
- LUT needs to be built when converting 24-bit colour images to 8-bit: grouping similar colours (each group assigned a colour entry)
- Possible for palette animation by changing the colour map



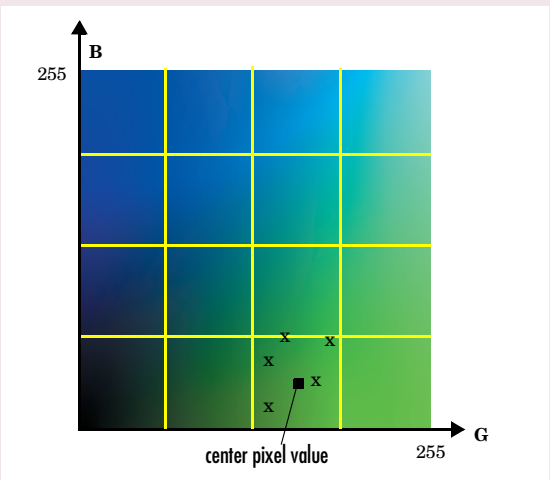
Colour LUT for 8-bit Colour Images

Converting Between 24-bit and 8-bit Colour Images

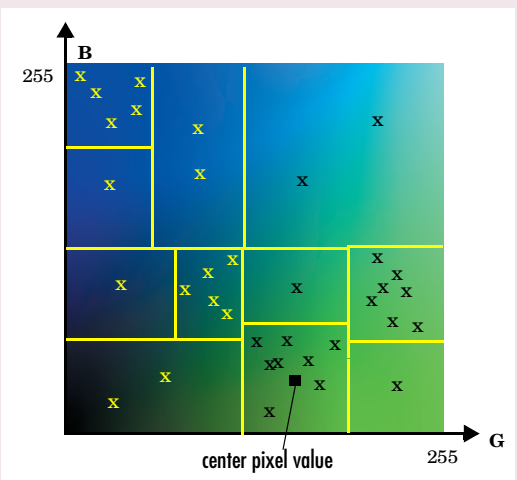
Approaches to Conversion

Dithering — as above but dither for **each** RGB channel **separately**.

Colour Quantisation — **Cluster** colours to the most common 256 colours, *E.g.*:



Uniform Quantisation



Minimum Variance Quantisation

Reading/Writing Images in MATLAB

imread()

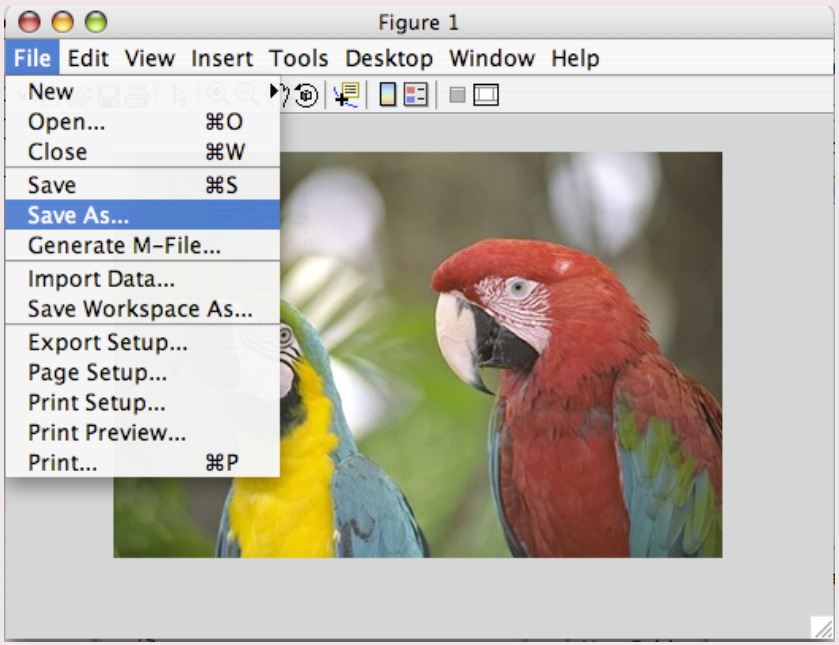
- `imread('filename')` reads a file and assigns it to an array variable:
  `im = imread('parrots.jpg');`
- MATLAB can read many file formats including: JPEG, GIFF, TIFF, BMP, PNG — **see** `help imread`
- This is a colour image so it has 3 images planes (RGB) — Its actually a 3-D array in MATLAB.
- `size()` is useful to get the dimensions of the image:
  `[l m n] = size(im);`

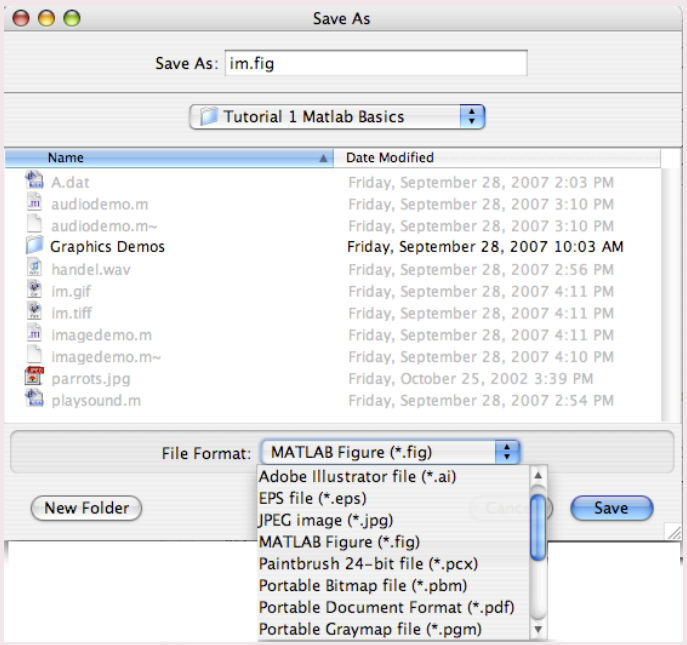
  Hint: can use `[l m ~] = size(im);` to save on variables.

Writing Images in MATLAB

imwrite()

- `imwrite()` write an image to a **specified** file format — **see** `help imwrite`
- Images (indeed any graphics) may also be saved directly from the MATLAB figure window.





Displaying Images in MATLAB

imshow()

- `im = imread('filename')` reads a file and assigns it to an array variable:

```
imshow(im);
```

MATLAB Image Code Example

```
%read image
>>im = imread('parrots.jpg');

% get image size (note 3 colours)
>>[l m n] = size(im)
l = 256
m = 384
n =  3

% get image size (note 3 colours)
>>[l m ~] = size(im)
l = 256
m = 384

% output as tiff image
>>imwrite(im,'im.tiff','tiff');

%display image
>>imshow(im);
```

Dithering in MATLAB

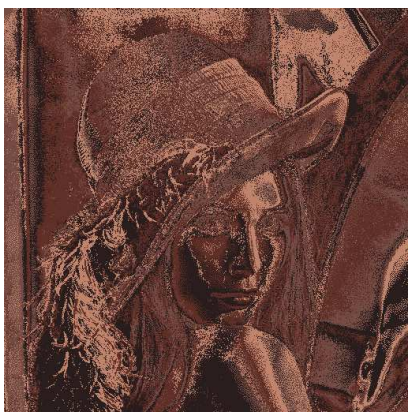
ditherbw_eg.m

```
% Load Image and Show
im = imread('lenabw.BMP');
imshow(im)

% Dither Image and Show Result
figure('Name', 'Dithered Image');
imdith = dither(im);
imshow(imdith);
```

24-bit to 8-bit Image Conversion

ditherrgb_eg.m

```
% Load Image and Show
im = imread('lenargb.bmp');
imshow(im)

% Dither Image and Show Result

[imdith,map] = rgb2ind(im,255);

figure('Name', '256 Colour Indices');
imshow(imdith);
colormap(map)

% Dither Image and Show Result

[imdith,map] = rgb2ind(im,32);

figure('Name', '32 Colour Indices');
imshow(imdith);
colormap(map)
```



Video

Video Basic Format

At the simplest level we can regard **video** as a sequence of images or **frames**:

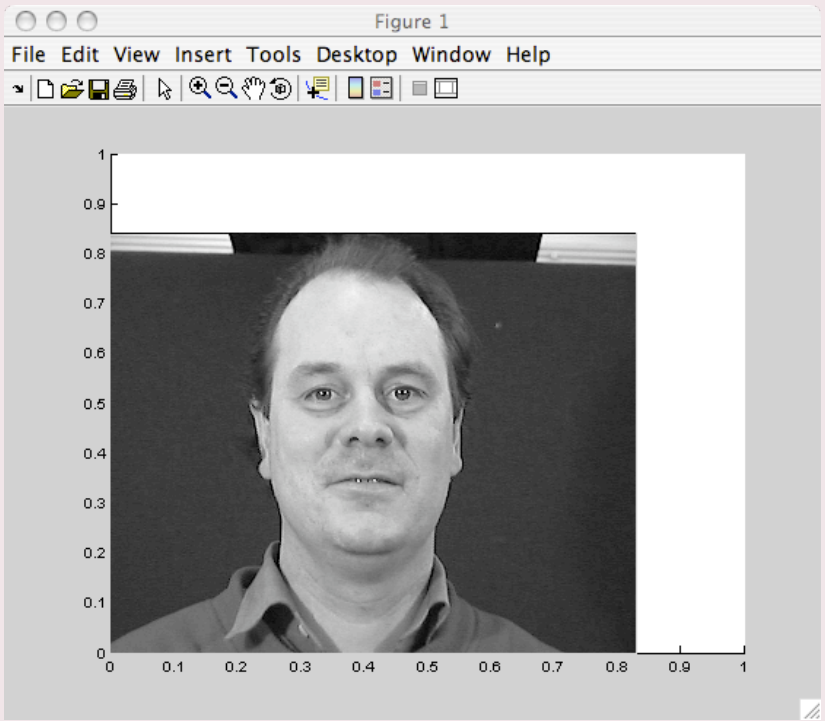
- Each frame sampled over time — typically around 25 to 30 frames per second (**Frame Rate**)

Simple Video Image Processing

- Extract each frame and process it.

Videos as Matrices in MATLAB

- MATLAB has a `movie` structure: it is a **2D array** of **image frames** over **time** (frame rate)
- MATLAB can natively read AVI video files with `movie_array = aviread('avi_file')` it returns a `movie`
- `movie(movie_array)` will play the video in a figure.



Processing Video with mmreader

mmreader()

- mmreader is a **powerful** tool to deal with **video** in MATLAB.
  - It supports more formats other than **AVI(.avi)**, e.g. **MPEG-1(.mpg)**, **Windows Media Video(.wmv, .asf, .asx)** on Windows, and **MPEG-1(.mpg)**, **MPEG-4(.mp4, .m4v)**, **QuickTime Movie(.mov)** on Mac. More details of the video can also be obtained.

An example:

- Create an mmreader object:

```
mmr = mmreader('origdave.avi');
```

- Read in all the frames

```
vidFrames = read(mmr);
```

  - For **true-colour video**, vidFrames is a **four dimensional array**: Width $\times$ Height $\times$ Channels (3) $\times$ numberOfFrames

Processing Video with mmreader(cont.)

Obtaining Frames

- **Obtain** the **number** of frames:

```
numFrames = get(mmr, 'numberOfFrames');
```

- Construct a movie struct to hold the video data.

```
for k = 1:numFrames
    mov(k).cdata = vidFrames(:, :, :, k);
    mov(k).colormap = [];
end
```

- Play the movie.

```
movie(mov, 1, get(mmr, 'frameRate'));
```

Aliasing in Images and Video

Nyquist Sampling Theorem Applies to Images and Video

It manifests itself in a few ways:

Spatial Aliasing — a function of the discretisation/sampling in the x and y dimensions.

Temporal Aliasing — (For video) a function of sampling at the frame rate.

Interlacing + Raster Scan Aliasing — A function of how video is captured or represented.

Aliasing in Images (1)

Aliasing Artifacts in Images

Stair-stepping — Stepped or jagged edges of angled lines, *e.g.*, at the slanted edges of letters.

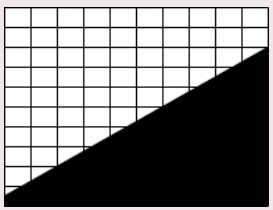
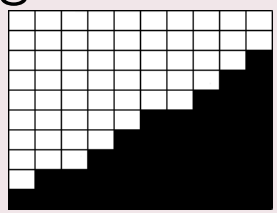



Image Zooming — changing resolution or not acquiring image in adequate resolution, *e.g.* digital zoom on cameras, digital scanning. (see zoom_alias.m)


Explanation: Simply Application of Nyquist's Sampling Theorem: Zooming in by a **factor** *n* divides the **sample** resolution by *n*

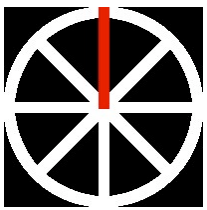
Aliasing in Images (1)

zoom_alias.m

```
im = imread('parrots.jpg');
zoom = 4

% get image dimensions
[n m o] = size(im)

% compute centre and window to zoom
nmid = n/2; mmid= m/2;
xoff = n/(zoom*2); yoff = m/(zoom*2);

% cut portion of image from centre
%proportional to zoom
newim = im(nmid-xoff:nmid+xoff,...
           mmid-yoff:mmid+yoff,:);

%zoom image use imresize function
newimzoom = imresize(newim,zoom);

% show images
figure(1), imshow(im);
title('Original Image')

figure(2), imshow(newim);
title('Original Image Cropped')

figure(3), imshow(newimzoom);
title('Zoomed Image (Aliasing Artifacts)')
```

Aliasing in Video: Temporal aliasing (1)

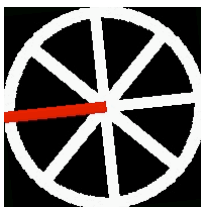
Temporal aliasing - e.g., rotating wagon wheel spokes apparently reversing direction, (see aliasing_wheel.m + spokesR.gif):


Frame 1


Frame2


Frame3


Frame4


Frame5


Below Nyquist Video


At Nyquist Video


Above Nyquist Video

Aliasing in Video: Temporal aliasing (2)

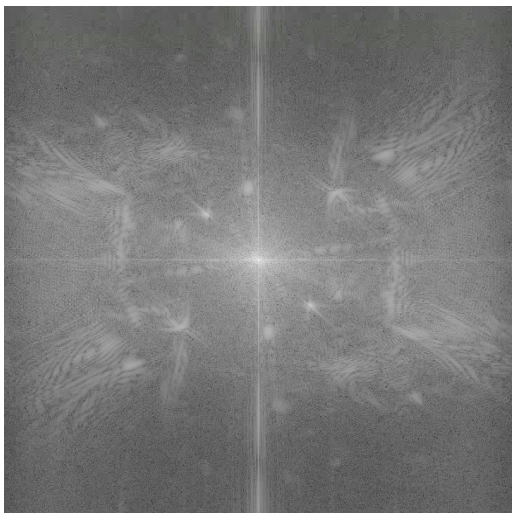
aliasing_wheel.m *fragment*

```
% Create Movie of rotating of wheel above sampling frequency
sampfreq = 15;

[im cmap] = imread('spokesR.gif');

[orign origm] = size(im);
offx = floor(orign/2);
offy = floor(orign/2);
rotfreq = 29;

rotstep = mod(360/(sampfreq/rotfreq),360)

movie_wheel = avifile('aliasing_wheel_oversampfreq.avi', 'fps', 2, ...
        'compression', 'none', 'colormap', cmap);

for i = 0:15
        rot = i*rotstep;
        IMR = imrotate(im,-1*rot);
        [n m] = size(IMR);
        centrex = floor(n/2);
        centrey = floor(m/2);
        IMR = IMR(centrex-offx +1:centrex+offx,centrey-offy + 1 :centrey+offy);
        movie_wheel = addframe(movie_wheel,IMR);
end;

movie_wheel = close(movie_wheel);
```

Aliasing in Video: Raster scan aliasing

Raster scan aliasing — e.g., twinkling or strobing effects on sharp horizontal lines, (see raster_aliasing.m + barbara.gif):

Strobing Alias Video



Strobing Alias Frequency Distributions Video

Aliasing in Video: Interlacing aliasing and Image Aliasing

Interlacing aliasing — Some video is interlaced, this effectively halves the sampling frequency. *e.g.*:
Interlacing Aliasing effects

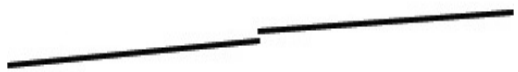


Image Aliasing — Stair-stepping/Zooming aliasing effects as images (as in Images above)

Explanation: Simply Application of Nyquist's Sampling Theorem