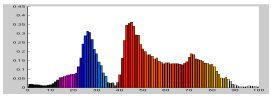
Dr. Qadri Hamarsheh



Digital Image Processing (750474)

Lecture 4

Outline of the Lecture

- **Writing images**
- **Data Classes**

Writing images

- Images are written to disk using function **imwrite**, syntax:

  imwrite (f , 'filename')

- Filename must include the **extension** of the file.

  imwrite (f, 'filename', 'extension')

Examples:-

```
>> imwrite (f, 'patient10_run1.tif ')
```

or alternatively

```
>> imwrite (f, 'patient10_run1, `tif')
```

- If filename contains no path information, then **imwrite** saves the file in the current working directory.

Special cases for imwrite syntax:

JPEG images:

For image with **JPEG** format, the syntax can have additional parameters:

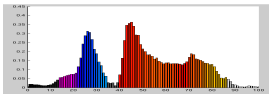
imwrite (f, 'filename.jpg', 'quality', q)

q: is an integer between 0 and 100 (the lower the number the higher the degradation due to **jpeg compression**).

Example:

- To reduce **storage** and **transmission** time, it is important that the images be compressed as much as possible while not degrading their visual appearance beyond a reasonable level (False contouring, for example).

```
>> f = imread('football.jpg');
>> imwrite (f,'football0.jpg','quality',0);
>> imwrite (f,'football5.jpg','quality',5);
>> imwrite (f,'football15.jpg','quality',15);
>> imwrite (f,'football25.jpg','quality',25);
>> imwrite (f,'football50.jpg','quality',50);
>> f0 = imread('football0.jpg');
>> f5 = imread('football5.jpg');
>> f15 = imread('football15.jpg');
>> f25 = imread('football25.jpg');
>> f50 = imread('football50.jpg');
>> subplot 231; imshow(f);title('Original Image');
>> subplot 232; imshow(f0); title('Image with q=0');
```



```
>> subplot 233; imshow(f5); title('Image with q=5');
>> subplot 234; imshow(f15); title('Image with q=15');
>> subplot 235; imshow(f25); title('Image with q=25');
>> subplot 236; imshow(f50); title('Image with q=50');
```

- **Notes about the program**

Quality value	Remark
q= 5 or q=0	quite visible(not suitable)
q= 15	false contouring visible
q= 25	acceptable solution with some error

- Some of the image/ graphics formats supported by imread and imwrite, starting with matlab 6.5.

TIFF, JPEG, GIF, BMP, PNG, XWD

  - **GIF** is supported by imread, but not by imwrite.

imfinfo function:

- The imfinfo function returns information about the image file, such as **size**, **width**, **height**, the **compression** achieved and other image file details.

Syntax:

imfinfo filename

```
>> imfinfo football25.jpg
```

The ouput of the imfinfo function

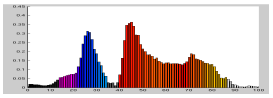
```
Filename: [1x93 char]
FileModDate: '27-Feb-2012 22:59:15'
FileSize: 5002
Format: 'jpg'
FormatVersion: ''
Width: 320
Height: 256
BitDepth: 24
ColorType: 'truecolor'
FormatSignature: ''
NumberOfSamples: 3
CodingMethod: 'Huffman'
CodingProcess: 'Sequential'
Comment: {}
```

- From the output of this command, we remark that:

File size (in bytes): the number of bytes in the original image is computed (original size):

$$\textbf{File size} = (\text{Width} * \text{Height} * \text{BitDepth}) / 8. \qquad (1)$$

$$\textbf{Compression ratio} = \frac{\text{file size (original image)}}{\text{file size (imfinfo result)}} \qquad (2)$$



- The **information** fields displayed by **imfinfo** can be captured into a structure variable that can be used for computations.

```
>> k= imfinfo ('football25.jpg');
```

k – **Structure name**; **k**- contains **fields**, that can be accessed by **dot** operator.

For example, the image height and width are stored in structure fields **k.Heigh** and **k.Width**.

Example using structure:

- Compute the **compression ratio** for football25.jpg:

```
>> K= imfinfo ('football25.jpg');
>> image_size = K.Width * K.Height * K.BitDepth/8;
>> compression_ratio = image_size/ k.FileSize
```

Compression_ratio = 49.1323

TIF images with imwrite

✓ The general **imwrite** syntax applicable for **tif** image;

Syntax:

```
imwrite (g,'filename.tif','compression','parameter','resolution',[colres rowres])
```

✓ **Parameter** can have one of the following values:

'none' – no compression.

'packbits' – pack bits compression (the default for **nonbinary** image).

'ccitt' – ccitt compression (the default for **binary** image).

The **1*2 array [colres rowres] –** (two integer), contains the column resolution and row resolution in **dots-per-unit** (the default values are [72 72]).

For example:- dpi=dots(pixels) per inch.

colres – dpi in **vertical** direction, **rowers**- dpi in **horizontal** direction.

✓ Specifying the resolution by single scalar, **res** is equivalent to writing **[res res].**

Example:-

- Suppose an 8-bit x-ray image of a circuit board generated during quality inspection. It is in **jpg** format, at **200 dpi**, the image is of size **450*450** pixels, so its dimensions are **2.25*2.25 inches**.

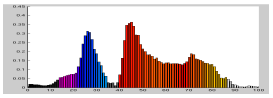
Question: store the image in **tif** format, with **no compression**, under the name **sf**, in addition we want to **reduce** the size of the image to 1.5*1.5 inches while **keeping** the pixel count at 450 * 450.

Solution:

```
>> imread(f,'xray.jpg);
>> res = round (200*2.25/1.5);
>> imwrite(f,'sf.tif','compression','none','resolution',res);
```

✓ The values of the vector **[colres rowres]** were determined by multiplying **200** dpi by the ratio. **2.25/1.5**.

✓ **round** function rounds its argument to the **nearest** integer n.



Data Classes

- The data classes supported by **matlab** and **IPT** for representing pixel values as in the following table:-

Type	Class Name	Description	Range	Bytes/ per element
numeric	double	Floating-point numbers	$(-10^{308}\ -10^{308})$	8
	uint8	Unsigned 8-bit integers	(0-255)	1
	uint16	Unsigned 16-bit integers	(0-65535)	2
	uint32	Unsigned 32-bit integers	(0-4294967295)	4
	int8	Signed 8-bit integers	(-128 -127)	1
	int16	Signed 16-bit integers	(-32768,32767)	2
	int32	Signed 32-bit integers	(-2147483648-2147483647)	4
	single	Floating point numbers	$-10^{38}\ -10^{38}$	4
Character	char	Characters, Unicode		2
Boolean	Logical	logical	0 or 1	1

- The frequently data classes that encountered in image processing are **double**, **uint8** and **logical**.
- Logical arrays are created by using function **logical** or by using **relational operators**.