FPT Education

FPT UNIVERSITY

PART I IMAGE PROCESSING

INSTRUCTOR: DR. NGUYEN NGOC TRUONG MINH

SCHOOL OF ELECTRICAL ENGINEERING, INTERNATIONAL UNIVERSITY (VNU-HCMC)

Ho Chi Minh City, June 2023

LECTURE III – IMAGE PROCESSING BASICS

INSTRUCTOR: DR. NGUYEN NGOC TRUONG MINH

SCHOOL OF ELECTRICAL ENGINEERING, INTERNATIONAL UNIVERSITY (VNU-HCMC)

Ho Chi Minh City, June 2023

LECTURE CONTENT

- What is MATLAB and why has it been selected to be the tool of choice?
- What programming environment does MATLAB offer?
- How do I read an image from a file using MATLAB?
- What are the main data classes used for image representation and how can I convert from one to another?
- Why is it important to understand the data class and range of pixel values of images stored in memory?

LECTURE CONTENT

- How do I display an image using MATLAB?
- How can I explore the pixel contents of an image?
- How do I save an image to a file using MATLAB?
- Chapter Summary – What have we learned?
- Problems

3.1 INTRODUCTION TO MATLAB

- MATLAB (MATrix LABoratory) is *a data analysis, prototyping, and visualization tool* with built-in support for matrices and matrix operations, excellent graphics capabilities, and a high-level programming language and development environment.
  - ➢ MATLAB has become very popular with engineers, scientists, and researchers in both industry and academia, due to many factors, among them, the availability of rich sets of specialized functions—encapsulated in toolboxes—for many important areas, from neural networks to finances to image processing.

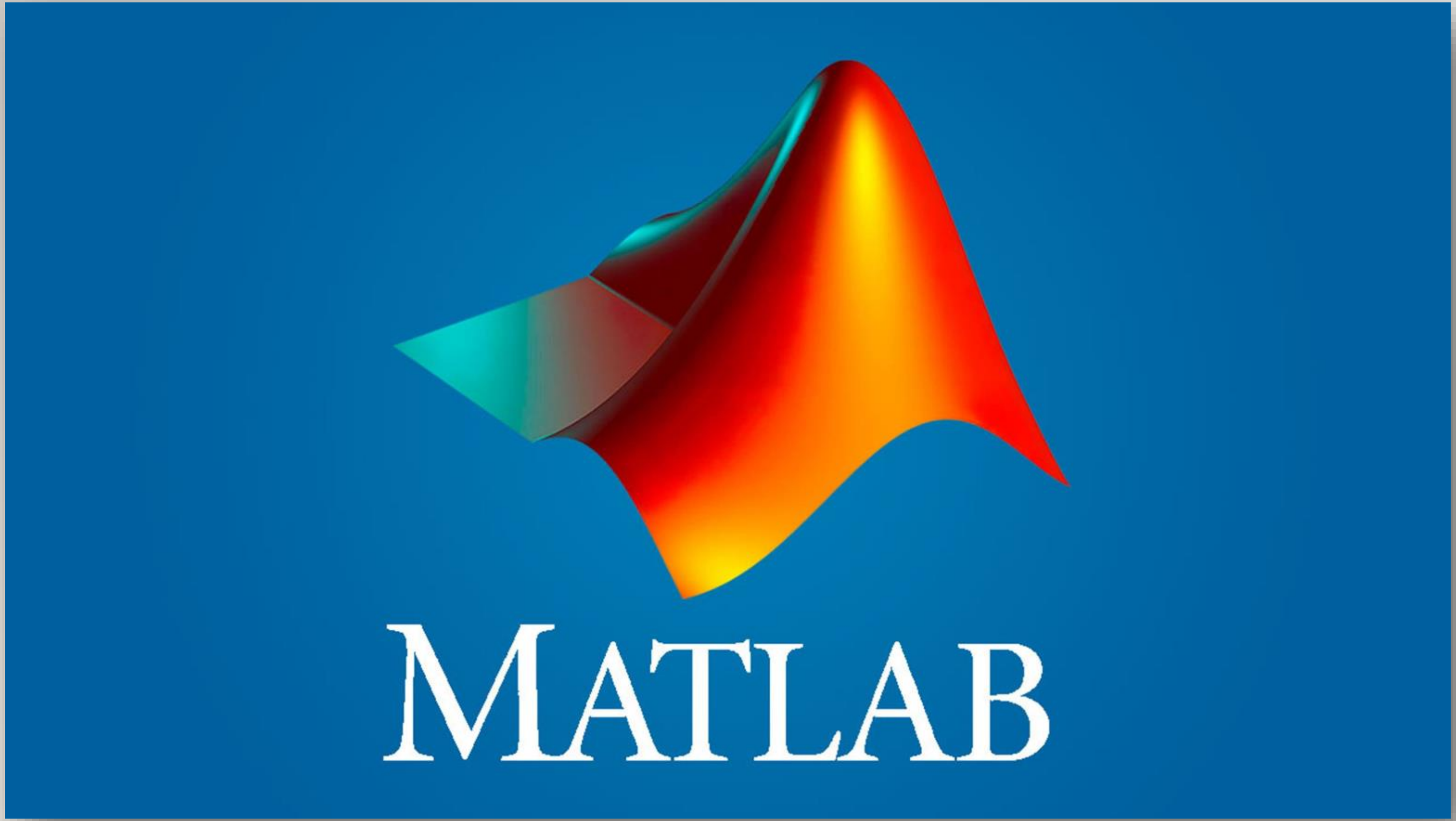
3.1 INTRODUCTION TO MATLAB

- MATLAB (MATrix LABoratory) is *a data analysis, prototyping, and visualization tool* with built-in support for matrices and matrix operations, excellent graphics capabilities, and a high-level programming language and development environment.
  - MATLAB has an extensive built-in documentation. It contains descriptions of MATLAB's main functions, sample code, relevant demos, and general help pages. MATLAB documentation can be accessed in a number of different ways, from command-line text-only help to hyperlinked HTML pages.

3.1 INTRODUCTION TO MATLAB

- MATLAB's basic data type is *the matrix* (or *array*). MATLAB does not require dimensioning, that is, memory allocation prior to actual usage. All data are considered to be matrices of some sort. Single values are considered by MATLAB to be 1 x 1 matrices.
- The MATLAB algorithm development environment provides a *command-line interface (CLI)*, an interpreter for the MATLAB programming language, *an extensive set of numerical and string manipulation functions*, 2D and 3D plotting functions, and the ability to build *graphical user interfaces* (*GUIs*). The MATLAB programming language interprets commands, which shortens programming time by eliminating the need for compilation.

3.1 INTRODUCTION TO MATLAB



R2023a

R2022b

3.2 BASIC ELEMENTS OF MATLAB

TABLE 3.1 MATLAB Data Classes

Data Class	*Description*
uint8	8-bit unsigned integers (1 byte per element)
uint16	16-bit unsigned integers (2 bytes per element)
uint32	32-bit unsigned integers (4 bytes per element)
int8	8-bit signed integers (1 byte per element)
int16	16-bit signed integers (2 bytes per element)
int32	32-bit signed integers (4 bytes per element)

3.2 BASIC ELEMENTS OF MATLAB

TABLE 3.1 MATLAB Data Classes

Data Class	*Description*
single	single-precision floating numbers (4 bytes per element)
double	double-precision floating numbers (8 bytes per element)
logical	values are 0 (false) or 1 (true) (1 byte per element)
char	characters (2 bytes per element)

3.2 BASIC ELEMENTS OF MATLAB

- The first eight data types listed in the table are known as *numeric classes*. A numeric value in MATLAB is of class *double* unless specified otherwise.
- Conversion between data classes (also known as *typecasting*) is possible (and often necessary). A string in MATLAB is simply a 1 x n array of characters.
- MATLAB arrays (vectors and matrices) are indexed using a *1-based convention*. Therefore, $a(1)$ is the syntax to refer to the first element of a one-dimensional array a and $f(1,1)$ is the syntax to refer to the first element of a two-dimensional array, such as the top left pixel in a monochrome image f. The colon (:) operator provides powerful indexing capabilities.

3.2 BASIC ELEMENTS OF MATLAB

Standard Arrays in MATLAB

- MATLAB has a number of useful, built-in standard arrays:
  - ➢ *zeros(m,n)* creates an m x n matrix of zeros.
  - ➢ *ones(m,n)* creates an m x n matrix of ones.
  - ➢ *true(m,n)* creates an m x n matrix of logical ones.
  - ➢ *false(m,n)* creates an m x n matrix of logical zeros.
  - ➢ *eye(n)* returns an n x n identity matrix.

3.2 BASIC ELEMENTS OF MATLAB

Standard Arrays in MATLAB

- MATLAB has a number of useful, built-in standard arrays:
  - *magic(m)* returns a magic square of order m.
  - *rand(m,n)* creates an m x n matrix whose entries are pseudorandom numbers uniformly distributed in the interval [0, 1].
  - *randn(m,n)* creates an m x n matrix whose entries are pseudorandom numbers that follow a normal (i.e., Gaussian) distribution with mean 0 and variance 1.

3.2 BASIC ELEMENTS OF MATLAB

TABLE 3.2 MATLAB Array and Matrix Arithmetic Operators

Operator	*Name*	*MATLAB Function*
+	Array and matrix addition	plus(a,b)
-	Array and matrix subtraction	minus(a,b)
.*	Element-by-element array multiplication	times(a,b)
*	Matrix multiplication	mtimes(a,b)
./	Array right division	rdivide(a,b)
.\	Array left division	ldivide(a,b)
/	Matrix right division	mrdivide(a,b)
\	Matrix left division	mldivide(a,b)

3.2 BASIC ELEMENTS OF MATLAB

TABLE 3.2 MATLAB Array and Matrix Arithmetic Operators (cont.)

Operator	*Name*	*MATLAB Function*
.^	Array power	power(a,b)
.^	Matrix power	mpower(a,b)
.'	Vector and matrix transpose	transpose(a)
'	Vector and matrix complex conjugate transpose	ctranspose(a)
+	Unary plus	uplus(a)
-	Unary minus	uminus(a)
:	Colon	colon(a,b) or colon(a,b,c)

3.2 BASIC ELEMENTS OF MATLAB

TABLE 3.3 Examples of MATLAB Specialized Matrix Operations

Name	*MATLAB Operator or Function*
Matrix transpose	Apostrophe (') operator
Inversion	*inv* function
Matrix determinant	*det* function
Flip up and down	*flipud* function
Flip left and right	*fliplr* function
Matrix rotation	*rot90* function
Matrix reshape	*reshape* function
Sum of the diagonal elements	*trace* function

3.2 BASIC ELEMENTS OF MATLAB

TABLE 3.4 Specialized Arithmetic Functions Supported by the Image Processing Toolbox (IPT)

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 (element-by-element) or multiplies a constant times an image
imdivide	Divides two images (element-by-element) or divides an image by a constant
imabsdiff	Computes the absolute difference between two images
imcomplement	Complements an image
imlincomb	Computes a linear combination of two or more images

3.2 BASIC ELEMENTS OF MATLAB

TABLE 3.5 Relational Operators

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

TABLE 3.6 Logical Operators

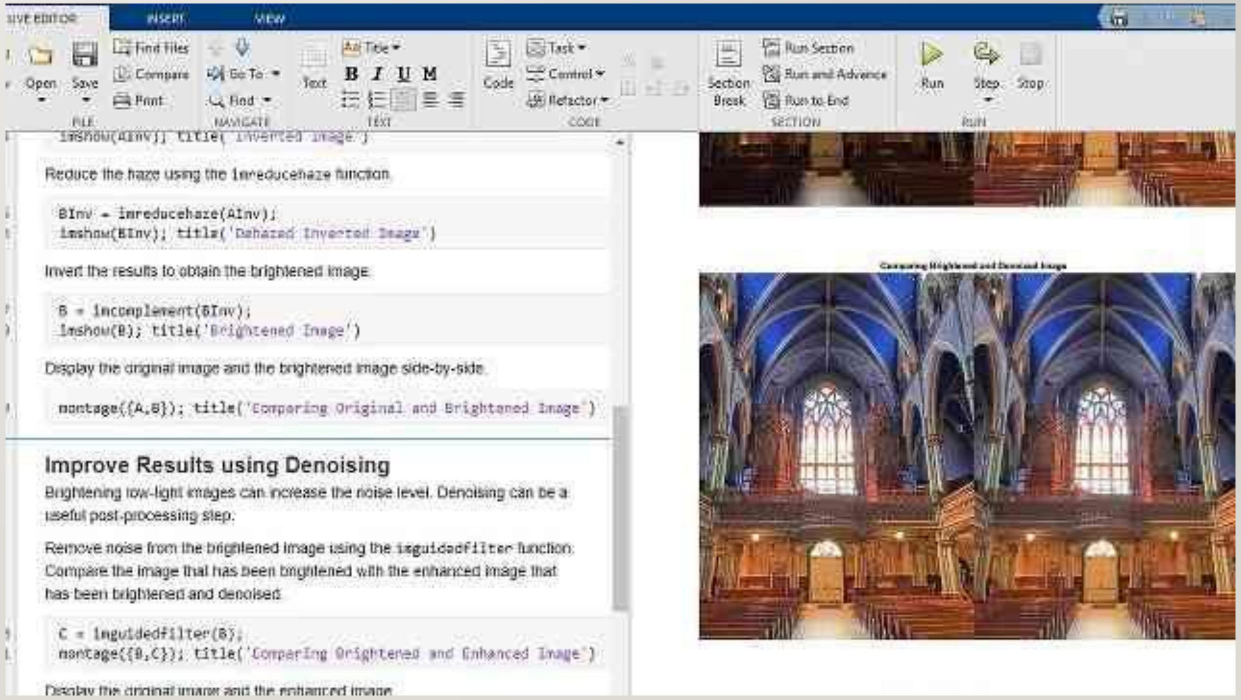
Operator	*Name*
&	AND
|	OR
~	NOT

TABLE 3.7 Logical Functions

Functions	*Description*
xor	Performs the exclusive-or (XOR) between two operands
all	Returns a1if all the elements in a vector are nonzero or a 0 otherwise. Operates column wise on matrices.
any	Returns a1if any of the elements in a vector are nonzero or a 0 otherwise. Operates column wise on matrices.

3.3 THE IMAGE PROCESSING TOOLBOX (IPT): AN OVERVIEW

- The Image Processing Toolbox (IPT) is a collection of functions that extend the basic capability of the MATLAB environment to enable specialized signal and image processing operations, such as the following:
- Spatial transformations
- Image analysis and enhancement
- Neighborhood and block operations
- Linear filtering and filter design
- Mathematical transforms
- Deblurring
- Morphological operations
- Color image processing



```
imshow(AInv); title('Inverted Image')
```

Reduce the haze using the imreducehaze function

```
BInv = imreducehaze(AInv);
imshow(BInv); title('Dehazed Inverted Image')
```

Invert the results to obtain the brightened image

```
B = imcomplement(BInv);
imshow(B); title('Brightened Image')
```

Display the original image and the brightened image side-by-side

```
montage({A,B}); title('Comparing Original and Brightened Image')
```

Improve Results using Denoising

Brightening low-light images can increase the noise level. Denoising can be a useful post-processing step.

Remove noise from the brightened image using the imguidedfilter function. Compare the image that has been brightened with the enhanced image that has been brightened and denoised.

```
C = imguidedfilter(B);
montage({B,C}); title('Comparing Brightened and Enhanced Image')
```

Display the original image and the enhanced image

3.4 ESSENTIAL FUNCTIONS AND FEATURES

Displaying Information About an Image File - *imfinfo*

imfinfo('maxresdefault.jpg');

```
Filename:           'C:\Users\MINH\Downloads\maxresdefault.jpg'
FileModDate:        '01-Apr-2021 06:53:18'
FileSize:           201014
Format:             'jpg'
FormatVersion:      ''
Width:              1280
Height:             720
BitDepth:           24
ColorType:  'truecolor'
FormatSignature:    ''
NumberOfSamples:    3
CodingMethod:       'Huffman'
CodingProcess:      'Sequential'
Comment:            {}
```

3.4 ESSENTIAL FUNCTIONS AND FEATURES

Reading an Image File - *imread*

The *imread* function allows you to read image files of almost any type, in virtually any format, located anywhere.

TABLE 4.1 IPT Functions to Perform Image Data Class Conversion

Name	*Converts an Image to Data Class*
im2single	single
im2double	double
im2uint8	unit8
im2uint16	uint16
im2int16	int16

3.4 ESSENTIAL FUNCTIONS AND FEATURES

Data Classes and Data Conversions

- The IPT ability to read images of any type, store their contents into arrays, and make them available for further processing and display does not preclude the need to understand how the image contents are represented in memory.
- The most common data classes for images are as follows:
  - *uint8*: 1 byte per pixel, in the [0, 255] range.
  - *double*: 8 bytes per pixel, usually in the [0.0, 1.0] range.
  - *logical*: 1 byte per pixel, representing its value as true (1 or white) or false (0 or black).

3.4 ESSENTIAL FUNCTIONS AND FEATURES

- Once the contents of an image have been read and stored into one or more variables, we are encouraged to *inspect the data class of these variables* and *their range of values* to understand how the pixel contents are represented and what is *their allowed range of values*.
- MATLAB allows data class conversion (*typecasting*) to be done in a straightforward way, but this type of conversion does not handle the range problem and is usually not what we want to do. To convert an image (or an arbitrary array for that matter) to a data class and range suitable for image processing, *we are encouraged to use one of the specialized functions listed in Table 4.1*. The input data class for any of those functions can be *logical*, *uint8*, *uint16*, *int16*, *single*, or *double*.

3.4 ESSENTIAL FUNCTIONS AND FEATURES

The IPT also contains other functions (not listed in Table 4.1) to convert the following:

- An image (of data class *uint8*, *uint16*, *single*, *int16*, or *double*) to a binary image (of data class *logical*) based on threshold: *im2bw*. Note that this is not simply a type (data class) conversion, but instead an implementation of a global thresholding algorithm.
- An image to an instance of the Java image class *java.awt.Image: im2java*
- An image to an instance of the Java image class *java.awt.image.BufferedImage: im2java2d*
- An image to a movie frame, *im2frame*
- A matrix to a grayscale image, *mat2gray*, where the input can be logical or any numeric class, and the output is an array of data class *double* (with values within the [0.0, 1.0] range).

3.4 ESSENTIAL FUNCTIONS AND FEATURES

- It is possible to convert a generic 2x2 array of class *double* (A) into its *uint8* equivalent (B) by simply typecasting it.

```
A =

     -8.0000      4.0000

           0      0.5000

>> B = uint8(A)

B =

           0      4

           0      1
```

The conversion consisted of truncation (all negative values became zero) and rounding off (0.5 became 1).

3.4 ESSENTIAL FUNCTIONS AND FEATURES

- Using *im2uint8* will lead to results within a normalized range ([0, 255]).

```
>> C = im2uint8(A)
C =
        0     255
        0     128
```

from which we can infer that it treated the original values as if they were in the [0, 1] range for data class *double*, that is, 0.5 became 128 (*midrange point*), anything less than or equal to 0 was truncated to 0, and anything greater than or equal to 1 was truncated to 255.

3.4 ESSENTIAL FUNCTIONS AND FEATURES

- Let us now apply *mat2gray* to A and inspect the results:

```
>> D = mat2gray(A)
D =
            0    1.0000
       0.6667    0.7083
```

- This result illustrates an important point: since A was already of data class *double*, there was no data class convention *per se*, but simply a range conversion—the smallest value (−8.0) became 0.0, the largest value (4.0) became 1.0, and all intermediate values were scaled within the new range.

3.4 ESSENTIAL FUNCTIONS AND FEATURES

- Finally, let us use *im2bw* to convert our images to *binary equivalents*.

```
>> E = im2bw(D, 0.4)
E =
         0     1
         1     1
```

- The results work as expected: any value greater than 0.4 becomes 1 (*true*), otherwise it becomes 0 (*false*).
- Note that since *im2bw* expects a threshold luminance level as a parameter and requires it to be a nonnegative number between 0 and 1, it implicitly assumes that the input variable (D, in this case) is a normalized grayscale image of class *double*. In other words, if you try to use A as an input image (*E = im2bw(A, 0.4)*), the function will work without any error or warning, but the result may not make sense.

3.4 ESSENTIAL FUNCTIONS AND FEATURES

- The IPT also includes functions to convert between RGB (*truecolor*), indexed image, and grayscale image, which are listed in Table 4.2.

TABLE 4.2 IPT Functions to Perform Image Data Class Conversion

Name	*Description*	
	Converts	**Into**
ind2gray	An indexed image	Its grayscale equivalent
gray2ind	A grayscale image	An indexed representation
rgb2gray	An RGB (truecolor) image	Its grayscale equivalent
rgb2ind	An RGB (truecolor) image	An indexed representation
ind2rgb	An indexed color image	Its RGB (truecolor) equivalent

3.5 DISPLAYING THE CONTENTS OF AN IMAGE

MATLAB has several functions for displaying images:

- *image*: displays an image using the current color map.
- *imagesc*: scales image data to the full range of the current color map and displays the image.
- *imshow*: displays an image and contains a number of optimizations and optional parameters for property settings associated with image display.
- *imtool*: displays an image and contains a number of associated tools that can be used to explore the image contents.

EXAMPLE 3.3

- The following MATLAB code is used to open an image file and display it using different *imshow* options:

```
I = imread('nature.jpg');
imshow(rgb2gray(I));
figure, imshow(rgb2gray(I),[]);
figure, imshow(rgb2gray(I),[100 160]);
```

- The first call to *imshow* displays the image in *its original state*. The following line opens a new figure and displays *a scaled* (for display purposes) *version of the same image*. The last line specifies *a range of gray levels*, such that all values lower than 100 will be displayed as black and all values greater than 160 will be displayed as white.

3.5 DISPLAYING THE CONTENTS OF AN IMAGE

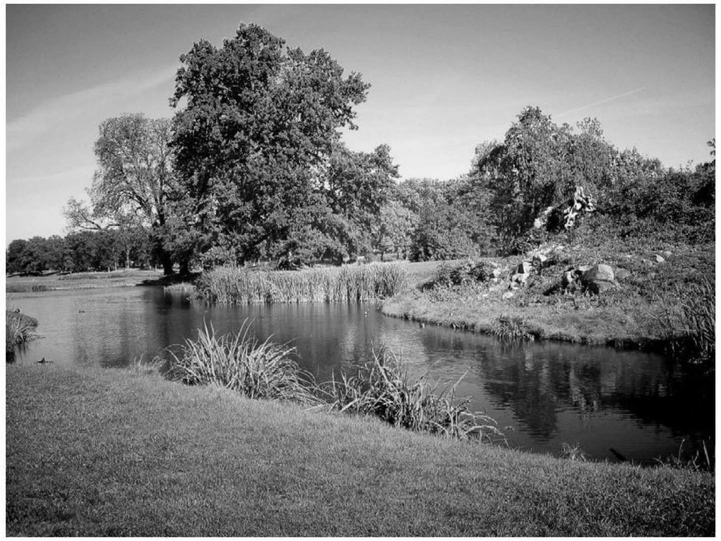
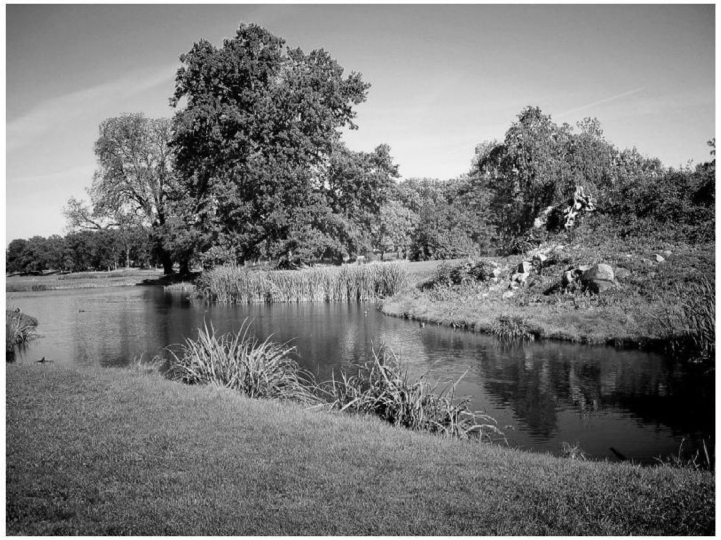
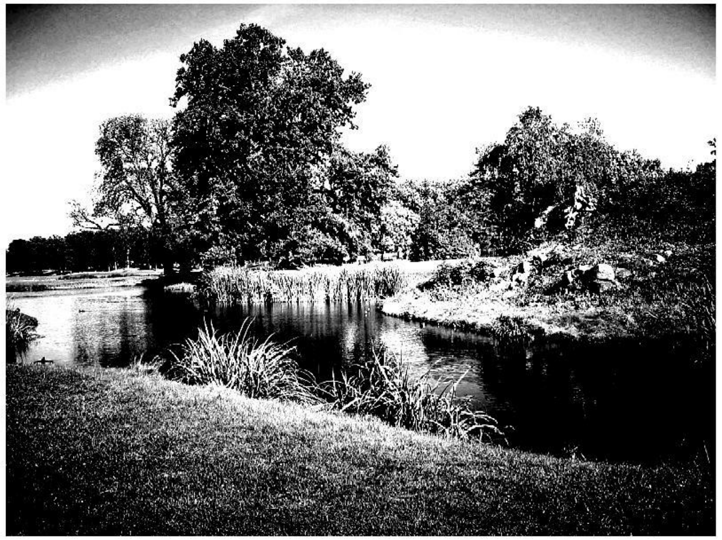


FIGURE 4.1 Displaying an image: (a) without scaling; (b) scaling for display purposes; (c) selecting only pixels within a specified range.

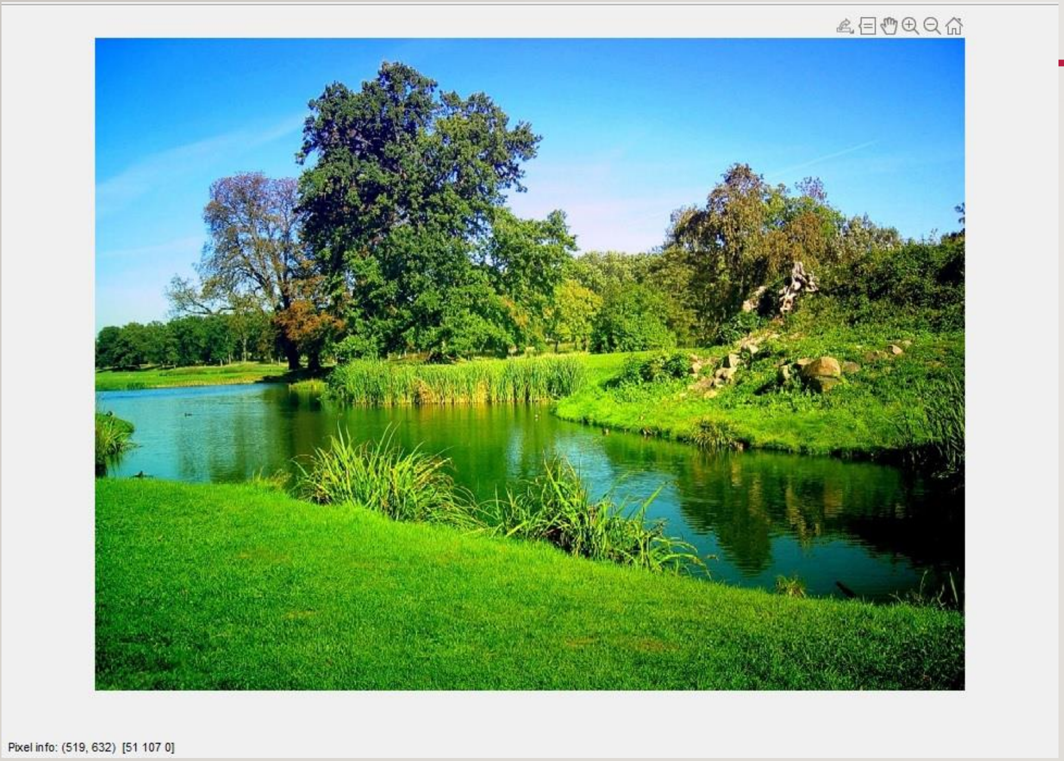




3.6 EXPLORING THE CONTENTS OF AN IMAGE

- Image processing researchers and practitioners often need to inspect the contents of an image more closely.
- In MATLAB, this is usually done using the *imtool* function, which provides all the image display capabilities of *imshow* as well as access to other tools for navigating and exploring images, such as the *Pixel Region* tool (Figure 4.2), *Image Information* tool (Figure 4.3), and *Adjust Contrast* tool (Figure 4.4).
- These tools can also be directly accessed using their library functions *impixelinfo*, *imageinfo*, and *imcontrast*, respectively.

3.6 EXPLORING THE CONTENTS OF AN IMAGE


Pixel info: (519, 632) [51 107 0]

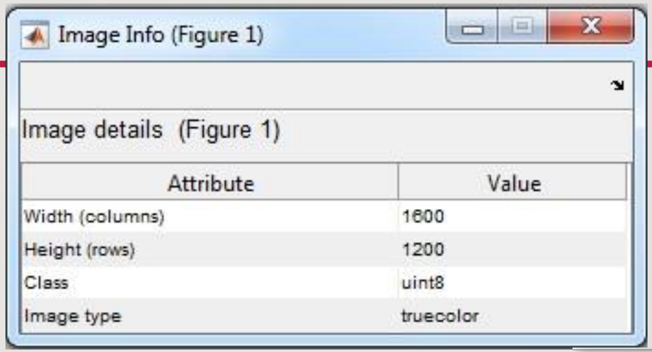


Image details (Figure 1)

Attribute	Value
Width (columns)	1600
Height (rows)	1200
Class	uint8
Image type	truecolor

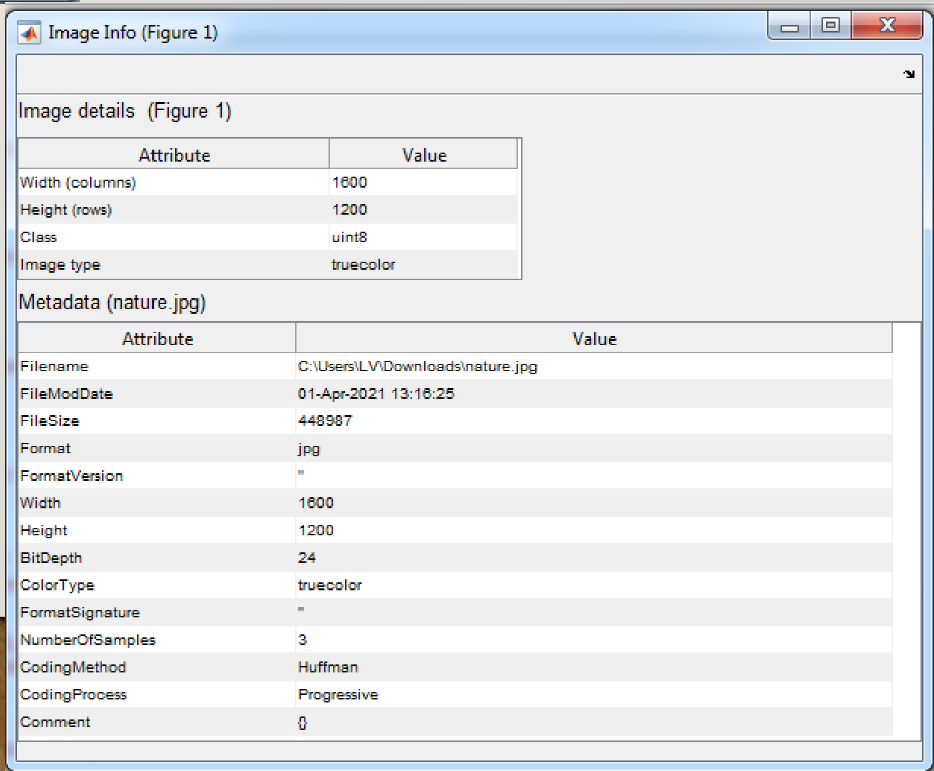


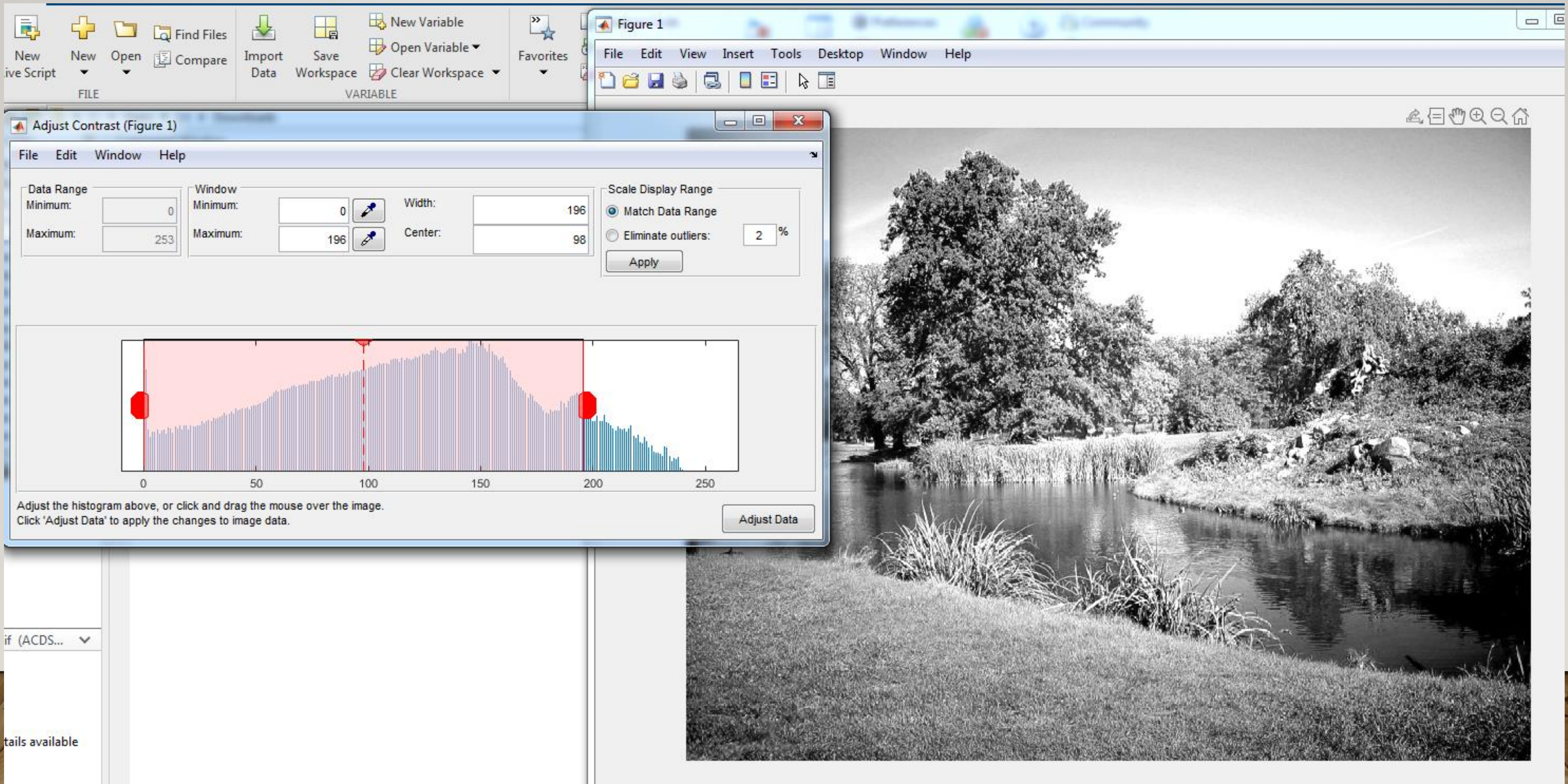
Image details (Figure 1)

Attribute	Value
Width (columns)	1600
Height (rows)	1200
Class	uint8
Image type	truecolor

Metadata (nature.jpg)

Attribute	Value
Filename	C:\Users\LV\Downloads\nature.jpg
FileModDate	01-Apr-2021 13:16:25
FileSize	448987
Format	jpg
FormatVersion	''
Width	1600
Height	1200
BitDepth	24
ColorType	truecolor
FormatSignature	''
NumberOfSamples	3
CodingMethod	Huffman
CodingProcess	Progressive
Comment	{}

3.6 EXPLORING THE CONTENTS OF AN IMAGE



Data Range
Minimum: 0
Maximum: 253

Window
Minimum: 0
Maximum: 196
Width: 196
Center: 98

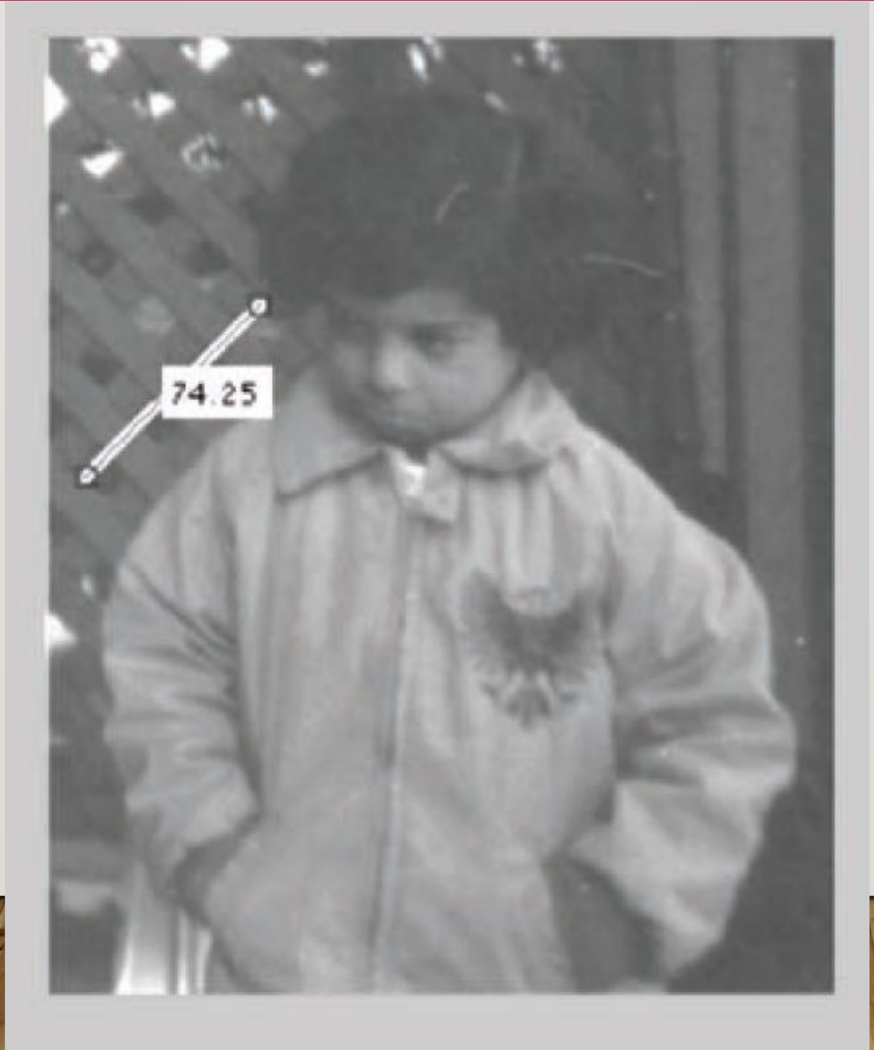
Scale Display Range
[✓ Match Data Range]
Eliminate outliers: 2 %

Adjust the histogram above, or click and drag the mouse over the image.
Click 'Adjust Data' to apply the changes to image data.

3.6 EXPLORING THE CONTENTS OF AN IMAGE

- Two other relevant IPT functions for inspecting image contents and pixel values are
  - *impixel:* returns the red, green, and blue color values of image pixels specified with the mouse.
  - *imdistline*: creates a Distance tool—a draggable, resizable line that measures the distance between its endpoints. We can use the mouse to move and resize the line to measure the distance between any two points within an image.

3.6 EXPLORING THE CONTENTS OF AN IMAGE



3.7 WRITING THE RESULTING IMAGE ONTO A FILE

- MATLAB's IPT has a built-in function, *imwrite*, to write the contents of an image in one of the most popular graphic file formats.
- If the output file format uses lossy compression (e.g., JPEG), *imwrite* allows the specification of a quality parameter, used as a trade-off between the resulting image's subjective quality and the file size.

```
I = imread('nature.jpg');
imwrite(I, 'naturedefault.jpg');
imwrite(I, 'nature05.jpg', 'quality', 5);
imwrite(I, 'nature95.jpg', 'quality', 95);
```

3.7 WRITING THE RESULTING IMAGE ONTO A FILE

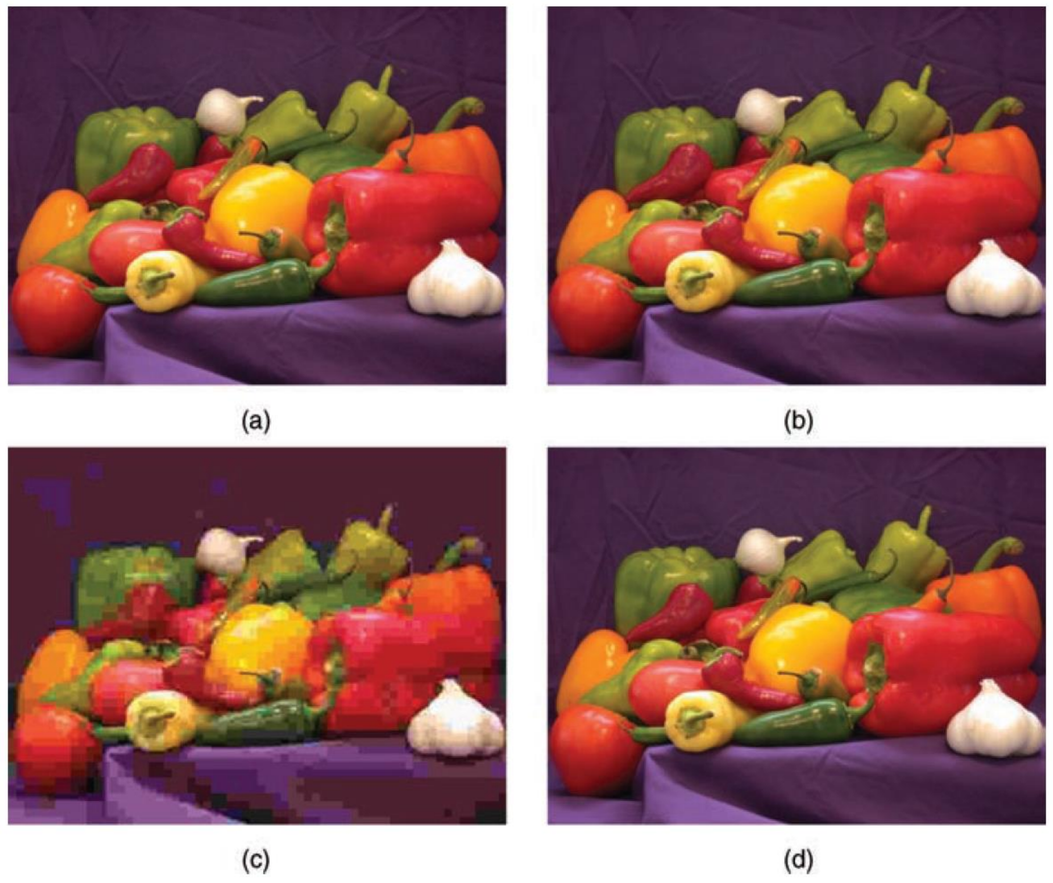


FIGURE 4.6 Reading and writing images: (a) Original image (PNG); (b) compressed image (JPG, $q = 75$, file size = 24 kB); (c) compressed image (JPG, $q = 5$, file size = 8 kB); (d) compressed image (JPG, $q = 95$, file size = 60 kB). Original image: courtesy of MathWorks.

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

Goal

- The goal of this tutorial is to explore basic image manipulation techniques using MATLAB and IPT.

Objectives

- *Explore the different image types supported by MATLAB and IPT.*
- *Learn how to read images into MATLAB.*
- *Explore image conversion.*
- *Learn how to display images.*
- *Learn how to write images to disk.*

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

1. Load the image coins.png by executing the following statement:

```
I = imread('coins.png');
```

Question 1: What type of image is coins.png?

Question 2: Why do we use the semicolon (;) operator after the imread statement? What happens if we omit it?

Binary, intensity, and truecolor images can all be read with the *imread* function as demonstrated above. When reading in an indexed image, we must specify variables for both the image and its color map. This is illustrated in the following step:

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

2. Load the image trees.tif.

```
[X,map] = imread('trees.tif');
```

Some operations may require us to convert an image from one type to another. For example, performing image adjustments on an indexed image may not give the results we are looking to achieve because the calculations are performed on the index values and not the representative RGB values. To make this an easier task, we can convert the indexed image to an RGB image using *ind2rgb*.

3. Convert the indexed image X with color map *map* to an RGB image, X_rgb.

```
X_rgb = ind2rgb(X,map);
```

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

Question 3: How many dimensions does the variable X_rgb have and what are their sizes?

4. Convert the indexed image X with color map map to an intensity image.

```
X_gray = ind2gray(X,map);
```

Question 4: What class type is X_gray?

5. We can verify that the new intensity image consists of pixel values in the range [0, 255].

```
max(X_gray(:))
min(X_gray(:))
```

Question 5: Why are we required to use the colon operator (:) when specifying the X_gray variable? What happens if we omit it?

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

It was demonstrated in the previous step that the *X_gray* image contained values in the range [0, 255] (in this particular image, they happened to be exactly 0 and 255, which is just a coincidence). Let us see what happens when we convert the image to class *double*.

6. Convert the variable X_gray to class double.

```
X_gray_dbl = im2double(X_gray);
```

Question 6: What is the range of values for the new variable X_gray_dbl?

Similarly, you can convert to other class types by using *im2uint8* and *im2uint16*, for example. When converting a *uint16* image to *uint8*, we must be careful because the conversion quantizes the 65,536 possible values to 256 possible values.

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

MATLAB comes with built-in image displaying functions. The image function can be used to display image data, and the *imagesc* function will perform the same operation but in addition will scale the image data to the full range of values. The IPT provides an enhanced image displaying function that optimizes settings on the image axes to provide a better display of image data: *imshow*.

7. Use the imshow function (with the impixelinfo option) to display the coins.png image that is currently loaded in the variable I.

```
imshow(I), impixelinfo
```

Binary, intensity, and truecolor images can all be displayed as demonstrated previously. To display indexed images, we must specify the color map along with the image data.

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

8. Display the indexed image *trees.tif*. The image data are stored in variable X and the color map in map. Note that the impixelinfo option provides a clear hint that this is an indexed color image.

```
imshow(X,map), impixelinfo
```

Question 7: Consider an image where the range of possible values for each pixel is not [0, 255], but a nonstandard range such as [0, 99]. How would we display the image so that a value of 99 represents white and a value of 0 represents black?

The *impixel* function allows the inspection of the contents of selected pixels of interest within the image.

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

9. Use the *impixel* function to explore interactively the pixel contents of selected points in the image. Use the mouse to click on the points of interest: normal button clicks are used to select pixels, pressing ***Backspace*** or ***Delete*** removes the previously selected pixel, a double-click adds a final pixel and ends the selection, and pressing ***Return*** finishes the selection without adding a final pixel.

```
RGB = imread('peppers.png');
[c,r,p] = impixel(RGB);
```

Question 8: What is the meaning of the values stored in variables r, c, and p?

The *improfile* function can be used to compute and plot the intensity values along a line or a multiline path in an image.

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

10. Use the *improfile* function to explore the contents of a line in the coins.png image that is currently loaded in the variable I.

```
r1 = 17; c1 = 18; r2 = 201; c2 = 286;
imshow(I);
line([c1, c2], [r1, r2], 'Color', 'g', 'LineWidth', 2);
figure;
improfile(I, [c1, c2], [r1, r2]);
ylabel('Gray level');
```

The *imtool* function is the latest and richest IPT function for displaying images. It provides all the image display capabilities of *imshow* as well as access to other tools for navigating and exploring images, such as *the Pixel Region tool*, *the Image Information tool*, and *the Adjust Contrast tool*.

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

11. Use the *imtool* function to display the image currently loaded in the variable *X_rgb*. Note that a secondary window (***Overview***) will open as well. Explore the additional functionality, including the possibility of measuring distances between two points within the image.

```
imtool(X_rgb)
```

We can display multiple images within one figure using the subplot function. When using this function, the first two parameters specify the number of rows and columns to divide the figure. The third parameter specifies which subdivision to use.

In the case of subplot(2,3,3), we are telling MATLAB to divide the figure into two rows and three columns and set the third cell as active. This division is demonstrated in Figure 4.7.

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

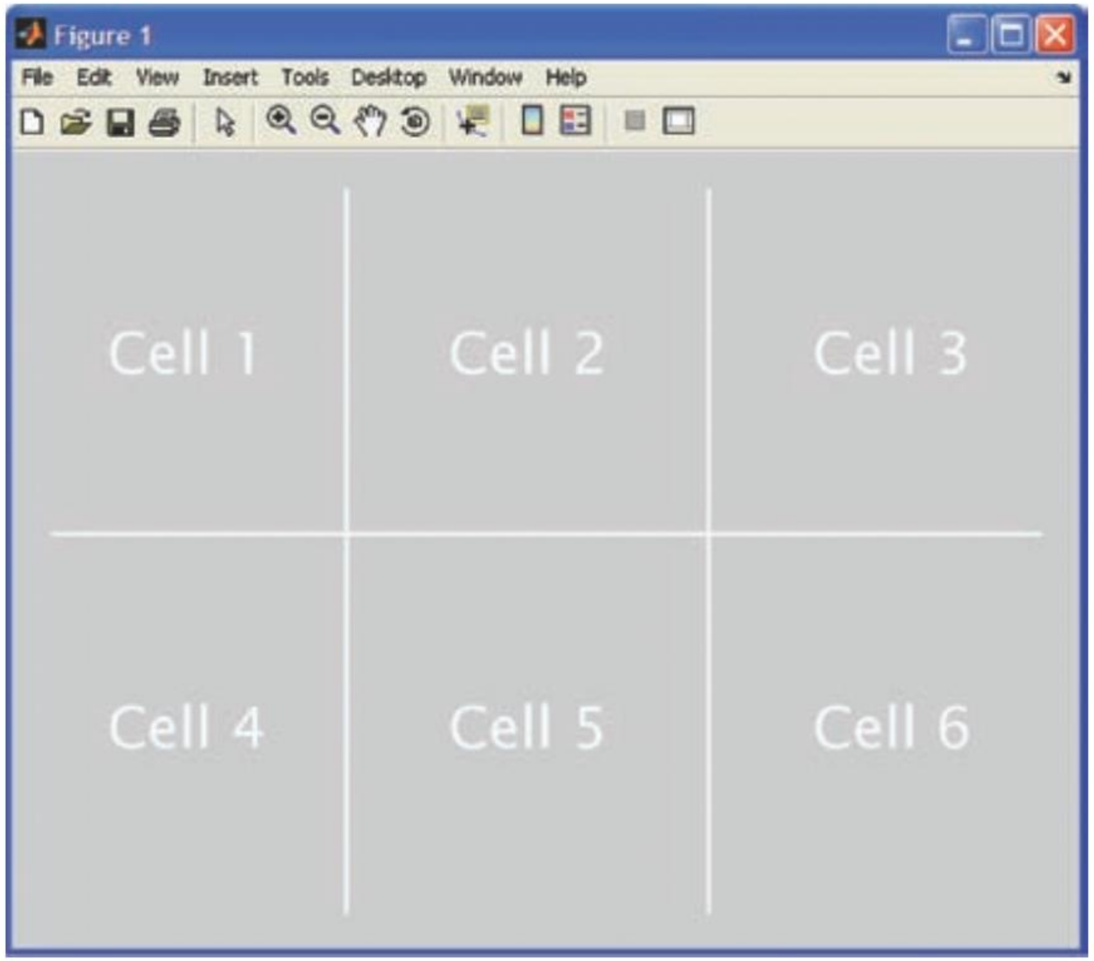


FIGURE 4.7 Division of a figure using subplot.

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

12. Close any open figures (close all).
13. Execute the following statements to create a subplot with two images:

```
A = imread('pout.tif');
B = imread('cameraman.tif');
figure
subplot(1,2,1), imshow(A)
subplot(1,2,2), imshow(B)
```

Question 9: What is the range of values for image A and image B?

In the previous step, we displayed two images, both of which were intensity images. Even though there is no color map associated with intensity images, MATLAB uses a grayscale color map to display an intensity image (this happens in the background and is usually invisible to the user). Let us consider the case where an intensity image and an indexed image are both displayed in one figure, using the subplot function as before.

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

14. Close any open figures.

15. Display the *coins.png* (loaded in variable I) and the *trees.tif* (loaded in variable X and its color map in variable map) images in a subplot. Execute each statement at a time to see the effect on the images as they are displayed.

```
figure;
subplot(1,2,1), imshow(I);
subplot(1,2,2), imshow(X,map);
```

Question 10: What happened to the coins image just after the trees image was displayed? Explain your answer.

To properly display images with different color maps, we must use the *subimage* function.

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

16. Use the *subimage* function to display multiple images with different color maps.

```
figure;
subplot(1,2,1), subimage(I), axis off;
subplot(1,2,2), subimage(X,map), axis off;
```

The *subimage* function converts the image to an equivalent RGB image and then displays that image. We can easily do this ourselves, but there is no direct conversion from intensity to RGB, so we must first convert from intensity to indexed, and then from indexed to RGB.

17. Manually convert the intensity image coins (loaded in the variable I) to an indexed image and then to RGB. Note that the trees image (loaded in variable X with its color map in variable map) has already been converted to RGB in step 3 (saved in variable X_rgb).

```
[I_ind,I_map] = gray2ind(I,256);
I_rgb = ind2rgb(I_ind,I_map);
```

3.8 TUTORIAL: BASIC IMAGE MANIPULATION

Question 11: What do the variables I_ind and I_map contain?

18. Display the truecolor images using the *imshow* function.

```
figure;
subplot(1,2,1), imshow(I_rgb);
subplot(1,2,2), imshow(X_rgb);
```

19. Use *imwrite* to save two of the modified images in this tutorial to files for further use. Use the JPEG format for one of them and the PNG extension for the other. For example,

```
imwrite(X_rgb, 'rgb_trees.jpg');
imwrite(X_gray, 'gray_trees.png');
```

WHAT HAVE WE LEARNED?

- MATLAB's IPT has a built-in function to open and read the contents of image files in most popular formats, *imread*.
- The most common data classes for images are *uint8* (1 byte per pixel, [0, 255]), *double* (8 bytes per pixel, [0.0, 1.0]), and *logical* (1 byte per pixel, *true*—white or *false*—black).
- Data class compatibility is *a critical prerequisite for image processing algorithms* to work properly, which occasionally requires data class conversions. In addition to standard data class conversion (*typecasting*), MATLAB has numerous specialized functions for *image class conversions*.
- A good understanding of the different data classes (and corresponding ranges of pixel values) of images stored in memory is *essential to the success of image processing algorithms*.

WHAT HAVE WE LEARNED?

- Ignoring or overlooking these aspects may lead to unwillingly truncating of intermediate calculation results, setting of wrong thresholds for pixel values, and many other potential problems.
- MATLAB has several functions for displaying images, such as *image*, *imagesc*, and *imshow*.
- To inspect the pixel contents of an image more closely, MATLAB includes the *imtool* function that *provides all the image display capabilities of imshow* as well as *access to other tools for navigating and exploring images*.
- To save the results of your image processing to a file, use MATLAB's built-in function *imwrite*.

PROBLEMS

Problem 1. Create a 2 x 2 array of type double $A = [1 \quad 4; 5 \quad 3]$; and write MATLAB statements to perform the following operations:

(a) Convert to a normalized ([0.0, 1.0]) grayscale image of data class *double*.

(b) Convert to a binary image of data class *logical*, such that any values greater than 2 (in the original image) will be interpreted as 1 (true).

(c) Repeat the previous step, this time producing a result of data class *double*.

Problem 2. The IPT function *gray2ind* allows conversion from a grayscale image to its indexed equivalent. What interesting property will the resulting color map (palette) display?

PROBLEMS

Problem 3. If you type help *imdemos* in MATLAB, you will see (among other things) a list of all the sample images that come with its IPT. Select five of those images and collect the following information about each of them:

- File name
- File format (extension)
- Type (binary, grayscale, truecolor, or indexed color)
- Size (bytes)
- Width (pixels)
- Height (pixels)

PROBLEMS

Problem 4. How does the IPT function *rgb2ind* handle the possibility that the original RGB image contains many more colors than the maximum palette (color map) size (65,536 colors)?

Problem 5. Select five images available in MATLAB (they may be the same as from the previous problem, or not, you choose). Open each of them using *imread*, save it (using *imwrite*) to (at least three) different file formats, and compare the resulting file size (in bytes) for each output format.

END OF LECTURE 3

LECTURE 4 – IMAGE SENSING AND ACQUISITION