Outline of the Lecture

- Image Types
- Converting between data classes and image types
- Converting images using IPT Function
- Matlab image Arithmetic Functions
- Array indexing

Image Types

- The toolbox supports four types of images:
  - Intensity Image.
  - Binary Images.
  - Indexed Images.
  - RGB Images.

Intensity Images: (Gray scale Images)

- An intensity image is a data matrix whose values have been scaled to represent intensities.

<table>
<thead>
<tr>
<th>Allowed Classes</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>0 - 255</td>
</tr>
<tr>
<td>uint16</td>
<td>0 - 65535</td>
</tr>
<tr>
<td>double</td>
<td>[0 - 1]</td>
</tr>
</tbody>
</table>

Intensity Images
**Binary Images:**

- Logical array containing only 0s and 1s, interpreted as **black** and **white**, respectively. In Matlab, by convention, **BW** is a variable Binary image.

**Binary Images**

- If the array contains 0s and 1s whose values are of data class different from logical (for example uint8), it is not considered a binary image in Matlab.

**Conversion a numeric array to binary:**

1. To convert, we use function `logical`.

```
>> x = [0 1 1 0 1 0];
>> y = logical (x);
```

If `x` contains other than 0s and 1s, the logical function converts all nonzero values to logical 1s.

2. Using relational and logical operators we can create a logical array.

- To test if an array is logical, we use `islogical` function:

```
>> islogical (y); % returns 1 if y is a logical array; otherwise it returns 0;
```

- `logical` array can be converted to numeric arrays using the data class conversion functions.

**Indexed Image**

- An indexed image consists of an array and a colormap matrix.
  - The pixel values in an array are direct indices into a colormap.
  - Each pixel has a value which does not give its color (as for an RGB image), but an index to the color in the map.
**By convention in matlab:**

- Variable \( X \) refer to array, variable \texttt{map} refer to the colormap.
- The array of class \texttt{logical, uint8, uint16, single} or \texttt{double}.
- The colormap matrix is an \( m \text{-by-} 3 \) array of class \texttt{double} (values in \([0 \ 1]\) range).
- Each row of \texttt{map} specifies the \texttt{red, green} and \texttt{blue} components of a single color.
- The color of each image pixel is determined by using corresponding value of \( X \) as an index into \texttt{map}.
- A colormap is often stored with an indexed image and is automatically loaded with image when using \texttt{imread} function.

### Relationship between values in the image matrix and the colormap

<table>
<thead>
<tr>
<th>class</th>
<th>Range of colormap</th>
</tr>
</thead>
<tbody>
<tr>
<td>single, double</td>
<td>1 through ( p ), ( p ) is the length of the colormap, Value 1- first row, 2 second, etc</td>
</tr>
<tr>
<td>logical, uint8 or uint16</td>
<td>value 0 points to the first row, value 1 points to the second, and so on</td>
</tr>
</tbody>
</table>

**RGB color Image (true color image)**

- True color images, require a three-dimensional array (\( m \text{-by-} n \text{-by-} 3 \)) of class \texttt{uint8}, \texttt{uint16}, \texttt{single} or \texttt{double} whose pixel values specify intensity values.

<table>
<thead>
<tr>
<th>class</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>single, double</td>
<td>[0,1]</td>
</tr>
<tr>
<td>\texttt{uint8}</td>
<td>[0 255]</td>
</tr>
<tr>
<td>\texttt{uint16}</td>
<td>[0 65535]</td>
</tr>
</tbody>
</table>

- Unlike an indexed image, however, these intensity values are stored directly in the image array, not indirectly in a colormap.
- \( m \) and \( n \) are the numbers of rows and columns of pixels in the image, and the third dimension consists of three planes, containing \texttt{red, green,} and \texttt{blue} intensity values.
- **For example:** to determine the color of the pixel \((112, 86)\)
  - Look at the RGB triplet stored in \((112, 86, 1:3)\). Suppose \((112, 86, 2)\) contains the value 0.1238, \((112, 86, 2)\) contains 0.9874 and \((112, 86, 3)\) contains 0.2543
  - The color of the pixel at \((112, 86)\) is: \(0.1238 \quad 0.9874 \quad 0.2543\)
Converting between data classes and image types

- Matlab expects operands in numeric computation to be of double.
- When you store an image, you should store it as uint8 image, since this requires far less memory than double.
- When you are processing an image, you should convert it to double, to convert, we use 2 methods:
  1) Type casting
     ✓ Convert from one data type to another: \( B = \text{data\_class\_name} \( (A) \)

Example (1):
\[
>> B = \text{double} \( (A) \)
\]
Example (2):
\[
>> D = \text{uint8} \( (c) \);
\]
  - If \( c \) is an array of class double, in which all values are \([0 \ldots 255]\) (possible fractional value).
  - If an array of class double has any values outside the range \([0 \ldots 255]\), matlab converts to 0 all values that are less than 0, and converts to 255 all values that are greater than 255.
  - Numbers in between are converted to integers by discarding their fractional parts.
  ✓ Converting any of the numeric data classes to logical
     - Results in an array with logical 1s in location where the input array has nonzero values and logical 0s where the input array contains 0s.

2) Converting between image classes and types.
   ✓ Perform necessary scaling to convert between image classes and types.
   a) \( \text{im2uint8} \( (x) \) \) detects input data class and scales to allow recognition of data as valid image data.

Example: Convert an image named \( x \) from double to uint8.
- Consider the following \( 2*2 \) image \( f \) of class double.
\[
>> f= [-0.5 \quad 0.5; 0.75 \quad 1.5]
\]
- Performing the conversion
\[
>> g= \text{im2uint8} \( (f) \)
\]
\begin{verbatim}
ans
 % g = 0 128
 % 191 255
\end{verbatim}

b) \( \text{im2double} \( (x) \) \) converts \( x \) input to class double in range \([0 1]\), unless input is of class double, no effect.

Example: Consider the class uint8 image.
\[
>> h = \text{uint8} \( ([25 \quad 50; 128 \quad 200])\);
\]
- Performing the conversion
\[
>> g = \text{im2double} \( (h) \);
\]
\begin{verbatim}
ans
 g= 0.0980 0.1961
 0.4706 0.7843
\end{verbatim}
c) \texttt{mat2gray (x, [Amin Amax])} takes arbitrary \texttt{double} array input and scaled to range \([0 1]\).

- \texttt{Values < Amin} function converts them to zero.
- \texttt{Values < Amax} function converts them to 1.

\[ \text{Convert an arbitrary array of double to an array of class double scaled to the range } [0 1]. \]

\[ \text{>> mat2gray (x)} \quad \% \text{sets the values of Amin and Amax to the actual minimum values in x.} \]

\[ \text{>> Im2bw (x, T)} \quad \text{converts intensity image (input matrix) to a binary image, anything less than } T \text{ output set to 0, otherwise output set to 1.} \]

\[ T = [0, 1] \]

\[ \text{Output is logical.} \]

\[ \text{Example: Convert the following double image } f = [1 \quad 2; \quad 3 \quad 4] \text{ to binary such that values 1 and 2 become 0 and the other two values become 1.} \]

\[ \text{Solution:} \]

\[ \text{First we convert it to the range } [0 1] \]

\[ \text{>> g= mat2gray (f)} \]

\[
\begin{array}{cc}
\text{ans} \\
g & 0 & 0.3333 \\
 & 0.6667 & 1.000 \\
\end{array}
\]

\[ \text{Then we convert it to binary using a threshold (0.6)} \]

\[ \text{>> gb= im2bw (g, 0.6)} \]

\[
\begin{array}{cc}
\text{ans} \\
gb & 0 & 0 \\
 & 1 & 1 \\
\end{array}
\]

Converting images using IPT Functions

<table>
<thead>
<tr>
<th>Matlab Command</th>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{dither (.)}</td>
<td>Gray scale to Binary images. RGB to Indexed images.</td>
</tr>
<tr>
<td>\texttt{gray2ind (.)}</td>
<td>Intensity to indexed images.</td>
</tr>
<tr>
<td>\texttt{ind2gray (.)}</td>
<td>Indexed to intensity images.</td>
</tr>
<tr>
<td>\texttt{ind2rgb (.)}</td>
<td>Indexed to RGB images.</td>
</tr>
<tr>
<td>\texttt{mat2gray (.)}</td>
<td>Regular matrix to intensity images, create a gray scale intensity image from data in a matrix by scaling the data.</td>
</tr>
<tr>
<td>\texttt{rgb2gray (.)}</td>
<td>RGB to intensity images.</td>
</tr>
<tr>
<td>\texttt{rgb2ind (.)}</td>
<td>RGB to indexed images.</td>
</tr>
<tr>
<td>\texttt{im2bw (.)}</td>
<td>Intensity to binary images.</td>
</tr>
</tbody>
</table>
Matlab image Arithmetic Functions

<table>
<thead>
<tr>
<th>Toolbox (IPT)</th>
<th>Matlab</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imadd (A,B)</td>
<td>A+B</td>
<td>Adding two images</td>
</tr>
<tr>
<td>imsubtract (A,B)</td>
<td>A-B</td>
<td>Subtracting two images</td>
</tr>
<tr>
<td>immultiply (A,B)</td>
<td>A.*B</td>
<td>Multiply two images</td>
</tr>
<tr>
<td>imdivide (A,B)</td>
<td>A./B</td>
<td>Divide two images (the values are rounded to the nearest integer, not truncated like true integer arithmetic)</td>
</tr>
</tbody>
</table>

Example: Reading a true color image into Matlab:

```
>> I = imread ('football jpg');
>> class (I) % uint8
>> size (I) % 250 320 3
>> figure
>> image (I);
>> title ('some title');
>> xlabel ('some text');
>> i(231,100,:)
    % ans (:, :, 1) = 48
    % ans (:, :, 2) = 37
    % ans (:, :, 3) =41
>> i= double (i) /255;
>> i (231, 100, :)
    % ans (:, :, 1) = 0.1882
    % ans (:, :, 2) = 0.1451
    %ans (:, :, 3) = 0.1608
>> class (i) % double
```

Array indexing

a) vector indexing

Examples:

```
>> v = [ 1 3 5 7 9] % row vector declaration
>> v(2)
% access the second element of the v.
>> w = v'
```

Dr. Quadri Hamarsheh
% row vector is converted to a column vector using 
% the transpose operator (')
>> v(1:3)
% To access blocks of elements, we use matlab's colon notation.
% Access first three elements of v.
>> v(2:4) % access the second through the fourth.
>> v(3:end) % all element from third to the last.
>> v(:) % produce a column vector.
>> v(1: end) % produce a row vector.
>> v(1:2:end) % start at 1, count up by 2 and stop when the count reaches the last.
% started at last, decreased by 2, and stopped at the first element.

b) Matrix indexing:
>> A = [1 2 3; 4 5 6; 7 8 9] % 3*3 matrix declaration.
>> A(2, 3) % access element in a matrix (2-row, 3-column).
>> C = A (:, 3) % (all rows, third columns).
>> T = A (1:2, 1:3) % extract the top two rows.
>> A(end, end) % Get the last element (last row, lost column).
>> E = A ([1 3], [2 3]); % using vectors to index into a matrix.