

# More on Images and Matlab

Prof. Eric Miller  
[elmiller@ece.tufts.edu](mailto:elmiller@ece.tufts.edu)



# Matlab Data Types

- Different means of representing numbers depending on what you want to do
- Examples:
  - Floating point numbers for scientific applications
    - 2.3543, -7.8956
  - Integers: 1, 2, -7, 6354, -2333430948
  - Unsigned integers for positive things (like pixel values): 255, 7, 0
  - True/false quantities (a.k.a. Boolean or logical) take values of 1 or 0.



# Representing numbers

- Computer memory arranges as set of bits organized into bytes.
  - Bit = zero or one
  - Byte = eight bits
- Each type of number is allocated a certain number of bytes
- 3 bit numbers integers
  - 3 locations:  $b_2b_1b_0$
  - Each  $b_i$  can be 0 or 1
  - Eight patterns: 000, 001, 010, 011, 100, 101, 110, 111
  - How to interpret? Many ways.
  - One way: integer between 0 and 7

$$x = \sum_{i=0}^2 b_i 2^i = b_0 + 2 \times b_1 + 4 \times b_2$$

- Examples:
  - 010  $\rightarrow 0 + 2 \times 1 + 4 \times 0 = 2$
  - 101  $\rightarrow 1 + 2 \times 0 + 4 \times 1 = 5$



# Representing numbers

- Which numbers do we represent? Depends on the type
- Unsigned int (**uint**) = 0 to 255
  - 00000000 = 0 ; 00000001 = 1;
  - 00000010 = 2 ; ... ; 11111111 = 255
- Signed integer (**int8**) = -128 to 127
  - 00000000 = -128 ; 00000001 = -127; ...
  - 01111111 = -1 ; 10000000 = 0 ; 10000001 = 1 ;
  - 10000010 = 2 ; ... ; 11111111 = 127
  - First bit kind of gives the sign of the number.
  - Remaining seven bits give the magnitude
- We need to tell the computer what we want to represent or we need to know how the computer defaults to some given representation

# Representing numbers

- Internal representation in computer a bit more convoluted than this
  - Based on something known as *two's complement*
  - Makes left bit really the sign bit (1 = negative and 0 = zero or positive)
  - Makes hardware-based arithmetic easy
  - Another way of interpreting the 256 patterns of zeros and ones
- Floating point
  - Really complicated story here. We'll not touch it.

# Images and Data Types

- **imread** in Matlab imports image information as different types:
  - **logical1** for binary
  - **uint8** for all all others
  - **double** for colormap
- Note that most of Matlab is based on doubles!!



# Displaying images

- **colormap**
  - Many rows by 3 column matrix (R, G, B columns)
  - Each column has doubles between 0 and 1 indicating intensity of each component
  - Grayscale images index into the rows
  - See help for list of built in colormaps (gray, winter, autumn, cool, hot, jet, ...)
- **image**
  - For grayscale, use **colormap** to set the colors
  - For color images, works as expected
- **imshow**
  - Works fine for **uint8** and **logical**
  - For binary images, make sure they are not **uint8**, but rather **logical**
  - For doubles, need to scale pixel values between 0 and 1

$$out = \frac{1}{max - min}(in - min)$$

# Bitplanes

- For 8 bit grayscale image, each pixel has eight binary components:

$$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$$

- Each bit is either 0 or 1
- The binary image formed by  $b_i$  for all the pixels is called the *i*-th *bit plane*
- See Matlab results
  - Why is  $b_7$  known as the *most significant bit* (MSB) and  $b_0$  the *least significant bit* (LSB)

# Bitplanes II

- Rather than using `bitget` to process the image pixels all at once we could visit each one individually
- Requires the use of a `for` loop in Matlab (see code)
- Loops are generally necessary coding structures for low level languages (C++, Java, Fortran, ...)
- Matlab can do many things in a “matrix” oriented way (e.g. `bitget`).
- While there is a loop someplace there, it is buried in compiled code.
- Explicit loops much slower in Matlab than taking advantage of built in matrix processing capabilities. Turns Matlab programming into something of a game.

# Bitplanes III

- Say extracting bitplanes was something we wanted to do over and over and over again.
- Pain to write the **for** loops every time
- Alternative: make a function that does the job for us
- Functions stored in Matlab **.m** files
- Example: function to find a specific bit plane in a given image
  - Note prolific use of comments and input error checking



# Matlab concepts covered

- Use of **whos** command to see contents of workspace
- Differences between **image** and **imshow**
- Use of **colormap** to provide false color to grayscale images
- Type casting with commands like **double**, **logical**, and **uint8**
- Use of **bitget** to extract bitplanes from **uint8** images
- Use of **for** loops
- Use of **isa** to find types
- Logical operators (**and**, **or**, **not**, **&**, **|**, **~**)
- Programming functions
  - Use of **error** to do input checking
  - Utility of comments



# Homework

- Can computers represent numbers like  $\pi$  or  $e$ ? Explain
- Why does **imshow** works so poorly for binary images that are of type **uint8**?
- Using **mod** and **floor**, rewrite **bitplane.m** to get rid of the for loops.
- Using **bitget**, rewrite **bitplane.m** to get rid of the for loops.
- Using **tic** and **toc**, determine the speeds of the three versions of **bitplane.m**