LAB3: Image Compression and Data Hiding

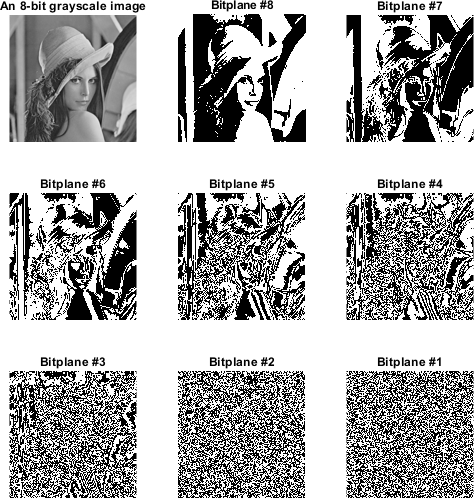
v4, revised on 2019-03-08 v3, revised on 2016-02-18; v2, revised on 2016-02-11; v1, created on 2015-10-20; by Prof. Min Wu ([minwu@umd.edu](mailto:minwu@umd.edu)) and Mr. Chau-Wai Wong ([cwwong@umd.edu](mailto:cwwong@umd.edu)); v4 by Chris Ayoub and Kevin Ho

# Contents

1. Bitplane Representation of Images
2. JPEG Image Compression
3. Image Hiding: An Intuitive Exploration Using GUI
4. Image Hiding: Write Your Own Program
5. Can Hidden Data Survive Image Compression? References

# Bitplane Representation of Images

As you discovered in prelab, a graylevel digital image is often represented by a matrix of unsigned 8-bit integers in which each location of the matrix corresponds to one pixel. One can take the -th bit  from each integer and form a new matrix with binary entries, and it is called the -th bitplane of the graylevel image; there are 8 of them for an 8-bit grayscale image. Below is an example showing an 8-bit grayscale image and all its bitplanes.



**Question 1**: Using the following skeleton code, generate the above subplot, including numbering of the titles. You will need to extract each bitplane using bitget, then display each bitplane with

imshow. Finally, title each image by using title with num2str and matrix concatenation. *Hint: Recall the prelab when using imshow - are your subplots showing a grayscale image or bitplanes?*

imgOri = imread('Lena128.bmp'); % read in image to imgOri variable

hFig = figure(2); set(hFig, 'position', [100 100 400 350]) % set the position and size of our figure

subplot(3,3,1); imshow(imgOri); title('An 8-bit grayscale image'); % display the original image in subplot 1 for k = 1 : 8

%TODO: extract bitplane into variable

subplot(3,3,10-k); %TODO: display the extracted bitplane into specified subplot; note that subplot(3,3,10-k) allows us to display the bitplanes in descending o

%TODO: title each subplot to show which bitplane is being displayed

end

truesize % adjust display size - call 'help truesize' in the command window to learn more.

# JPEG Image Compression

Lossy compression schemes such as JPEG are designed to achieve a high compression ratio while maintaining as much visual quality as possible. In Matlab, you can use the following function to compress an image using the JPEG compression scheme and save it to the hard disk with file name compressedImg.jpg:

## imwrite(img, 'compressedImg.jpg', 'Quality', qualityFactor)

where img is name of the variable pointing to the image data stored in MATLAB, and qualityFactor is a number between 0 to 100, with 100 the highest quality and 0 the lowest quality. You can use imread() to load a saved image file back into MATLAB. To check the file size of compressedImg.jpg in bytes, use

## fsize = getFilesize('compressedImg.jpg');

where variable fsize is used to store the returned file size.

Try JPEG compression on image Lena512.bmp by using a for loop to go through all quality factors stored in vector qualityFactor\_vector shown as follows:

## qualityFactor\_vector = [100:-10:10];

Use the getFilesize command to check the size of corresponding compressed files and put the returned size in the vector initialized as follows:

## filesize\_vector = zeros(size(qualityFactor\_vector));

Observe the visual quality of images compressed with different quality factors.

**Question 2**: In your for loop, save the images for quality factors 90%, 50%, and 10%. In a 1x4 subplot, display the original image along with the compressed images at these quality factors (90%, 50%, 10%). Title each figure to show which quality factor was used (you do not need to use num2str here).

**Question 3**: Describe the differences you see between the images compressed at the above quality factors.

**Question 4**: After finding the filesize for each quality factor, plot the file size vs. quality factor of the compressed image using the following two lines:

## stem(qualityFactor\_vector, filesize\_vector);

xlabel('quality factor (%)'); ylabel('filesize (bytes)'); title('filesize vs. image quality plot');

**Question 5**: What is the relationship between quality factor and file size?

# Image Hiding: An Intuitive Exploration Using GUI

In this part, you are going to hide or embed one image into another using an off-the-shelf program wrapped with a graphical user interface (GUI). Tune the parameters and observe different results. **In both parts of this secton, use kodim20.png as your host image and pepper.tiff as your embedded image. Check the 'Gray Mode' box for BOTH of these images.** Also note that the demo uses an offset in bitplane numbering scheme.

**To watch these videos**, click the links below to download a .swf (shockwave flash) file. You will need a flash player to view the demo (so please dig Internet Explorer out from the graveyard). Drag the

.swf file onto an Internet Explorer webpage to watch the demo. If you can figure out how to work the GUI without watching the videos, go for it.

### Image hiding without compression

Watch [this video (right-click and open a new window)](https://ece.umd.edu/%7Eminwu/course/EE101_S17/lab3/demo1.swf) showing the procedure of a binary image (more precisely, a bitplane of an image to be embedded) being embedded into a bitplane of a host image. Play with the *Image Hiding Demo* imgHidingMemo.m by changing the bitplanes of the host image and the image to be embedded at the "no compression" setting.

**Question 6**: Which are the optimal bitplanes for image hiding?

### Image hiding with compression

Watch [this video (right-click and open a new window)](https://ece.umd.edu/%7Eminwu/course/EE101_S17/lab3/demo2.swf) showing the procedure of a binary image being embedded into a bitplane of a host image. Focus on the extracted image after decompression.

Run the *Image Hiding Demo* program by calling imgHidingMemo.m. Change the bitplanes of the host image and the image to be embedded at various compression levels, and expect different results.

**Question 7**: Which are the optimal bitplanes for image hiding at given compression levels such as 90%, 50%, and 10% in your viewpoint?

**Question 8**: Is the above result dependent on images used? Why or why not?

# Image Hiding: Write Your Own Program

In this part, you are going to write your own piece of code that hides the bitplane of one image into a bitplane of another image. Essentially, you will write the code backing the demo your worked with in Part III. To simplify the programming steps, you will work on an example where the host image and the image to be embedded are of the same height and width.

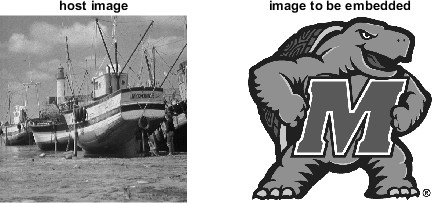
**Task**: You are given two grayscale images stored in variables img\_boat and img\_umdMark, and two parameters, hostImg\_bit and umdMark\_bit. Please write an embedding and the corresponding extraction code in Matlab to embed the bitplane of img\_umdMark as specified by the variable umdMark\_bit into the bitplane of img\_boat as specified by the variable hostImg\_bit. You are not required to write a MATLAB function, but it may be helpful. You should, however, make it easy to change which bitplane you extract and which bitplane you embed into.

The following code reads in our host and embed images and displays them.

img\_boat = imread('boat.tiff');

img\_umdMark = rgb2gray(imread('UMD\_mark\_Athletic.tiff')); figure(108);

subplot(1,2,1); imshow(img\_boat); title('host image'); subplot(1,2,2); imshow(img\_umdMark); title('image to be embedded');



### Step-1: Image Embedding

Use img\_boat as the host image and img\_umdMark as the image to be embedded.

**Question 8**: Write several lines of Matlab code to store the bitplane of img\_umdMark as specified by the variable umdMark\_bit into the bitplane of img\_boat as specified by the variable hostImg\_bit in order to obtain a composite image stored in the variable img\_composite. E.g., Start with the case umdMark\_bit = 8 and hostImg\_bit = 2. In other words place the 8th bitplane of img\_umdMark into the 2nd bitplane of img\_boat and store the output into img\_composite.

Hint: You may find functions bitget() and bitset() useful. Details of a Matlab function with the name func can be obtained in the Matlab console by typing the command help func.

### Step-2: Image Extraction

**Question 9**: Write a few lines of Matlab code to extract the bitplate as specified by the variable hostImg\_bit from the composite image img\_composite. As you did in Question 8, use the case

umdMark\_bit = 8 and hostImg\_bit = 2. In other words extract the 2nd bitplane from hostImg\_bit to acquire the 8th bitplane of umdMark\_bit (assuming your image embedding works).

**Question 10**: In a 1x3 subplot, show the original image, the composite image, and the extracted bitplane from Question 9

**Question 11**: Apply your Matlab code to do the embedding and extraction when the parameter pair (umdMark\_bit, hostImg\_bit) takes values of (8,1), (8,4), and (1,8), respectively. In other words, repeat Questions 8-10 for each of these pairs of paramters. Make sure to label each figure to show which parameter pair you used.

# Can Hidden Data Survive Image Compression?

**Question 12**: Try JPEG compression on composite images using the techniques of Questions 8-10 in Part IV by using three different quality factors 90, 50, and 10. For each quality factor, identify the lowest image bitplane for the embedded bitplane to survive under the current data embedding scheme.

In the above question you should find that you will lose Testudo at a low enough quality factor. We introduce the concept of redundancy to combat image compression in the rest of this section.

**Task**: To make a hidden bitplane more resilient against JPEG compression, one can repeat each binary pixel more than one time and then embed, and carry out a majority voting for each pixel after extracting the embedded data. In doing so, we are introducing redundancy before compression.

You are given a host image that has 4 times pixels of that of the image to be embedded. In doing so, you place your embedded image four times over in the host image. By placing our image 4 times over, we hope that the image compression does not affect every area the same - eventually we would like to reconstruct the embedded image by pooling together each of the four quadrants we had embedded.

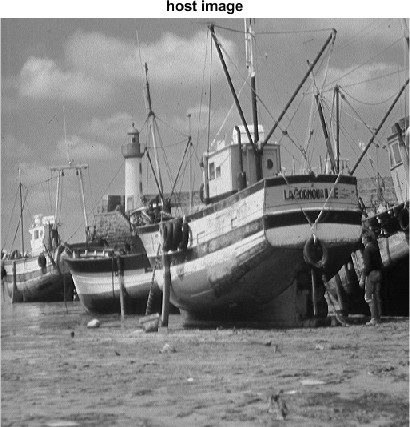
% the following lines of code use variables which store images which you

% loaded in during Part IV. img\_host = img\_boat;

img\_toBeEmbedded = imresize(img\_umdMark, 1/2, 'bicubic'); % reduce image size by 1/2 in both height and width

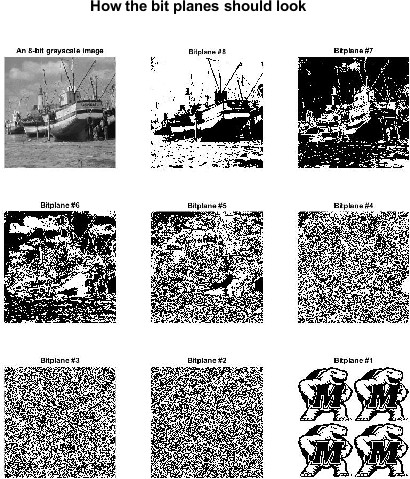
figure(109); imshow(img\_host); title('host image');

figure(110); imshow(img\_toBeEmbedded); title('image to be embedded');

**Question 13**: Modify your **Image Hiding** code developed in Section IV so that each pixel of the image to be embedded appears exactly 4 times in order to form a bitplane of the same height and width as those of the host image.

Your new images' bitplanes should resemble the ones below where one bitplane hold 4 copies of Testudo side by side. The example image uses the 8th bitplane of Testudo and hides it into the 1st bitplane of the boat.



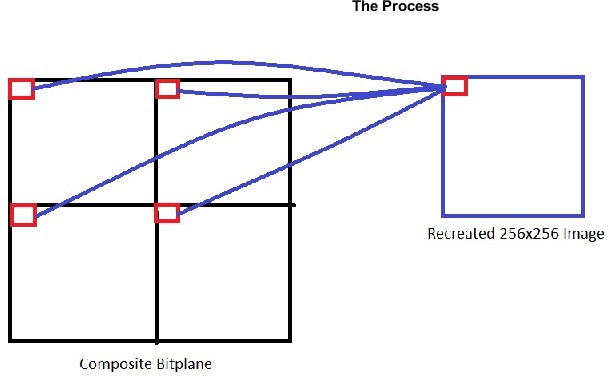
**Question 14**: After compressing your new image we will try to recreate the original testudo image that was hidden. You can imagine that your bit plane with the 4 Testudos can be split up into 4 quadrants. You will have add the corresponding pixel values in each quadrant together. For instance if we wanted to add the starting pixel in each quadrant we would add pixel (1,1) + (257,1)+(1,257)+ (257,257). Save this sum.

You will then use this sum **for each pixel** to recreate a 256x256 image.

if the sum is a 4 or 3, Then the value you put into the 256x256 image is a 1.

if the sum is a 0 or 1, Then the value you put into the 256x256 image is a 0.

if the sum is a 2, Then the value you put into the 256x256 image is a 0.5.



**Question 15**: Try JPEG compression on the new composite images by using three different quality factors 90, 50, and 10. For each quality factor, identify the lowest image bitplane for the embedded bitplane to survive under the error resilient embedding scheme.

# References

Matlab online tutorial: "MATLAB Onramp" https://matlabacademy.mathworks.com/R2015a/

ENEE408G course material: <http://www.ece.umd.edu/class/enee408g.F2006/image/>

[*Published with MATLAB® R2018b*](https://www.mathworks.com/products/matlab/)