Module 5 Lab 3: Bitplanes, Image Hiding, and Image Compression

Module by: Prof. Min Wu; Prelab by: Kevin Ho

**PRELAB**

***Directions:***

Read the following introduction to Module 5 Lab 3. Answer all five questions (which are bolded and underlined) in this Word Document. You DO NOT need to use MATLAB to complete this prelab – you may simply write your answers in Word.

***Introduction***:

In Module 5, Labs 1 and 2 we looked at various representations of images, including RGB, HSV, and grayscale. In Lab 3, we extend the use of grayscale images by introducing bitplanes. We’ll use bitplanes to introduce image compression and image hiding. The following example introduces the notion of a bitplane.

Imagine that we take a (very small) 2x2 pixel slice of an 8-bit GRAYSCALE image. Remember that for an 8-bit grayscale image, each pixel’s intensity can take values between 0-255. As an example, say our intensity values are:

Here we use the subscript 10 to denote that these are base-10 values.

We can represent this matrix using (8-bit) binary values as follows:

You DO NOT need to know the details of decimal-to-binary conversion for this lab, but if you’re curious you can read more about it [here](https://www.electronics-tutorials.ws/binary/bin_2.html).

We summarize and justify the decimal-to-binary conversion by showing how to convert from binary-to decimal-below. We will use 36 and 195 (in base 10) as our example.

(Unless otherwise denoted with a subscript, the numbers above are in base 10.)

In our conversion above, the leftmost bit (i.e. the leftmost binary digit) is the most significant bit (MSB), while the rightmost bit is the least significant bit (LSB). Notice here that, in terms of grayscale values, smaller numbers (closer to 0, i.e. black) will have MSB 0. Meanwhile, larger numbers (closer to 255, i.e. white) will have MSB 1.

**Question 1:** Why do we consider the leftmost bit to be the most significant bit? Why do we consider the rightmost bit to be the least significant bit?

The leftmost bit has the greatest effect on the number generated from the binary, whereas the rightmost bit is least significant because it has the least effect on the number generated from the binary.

|  |  |
| --- | --- |
|  |  |
|  |  |

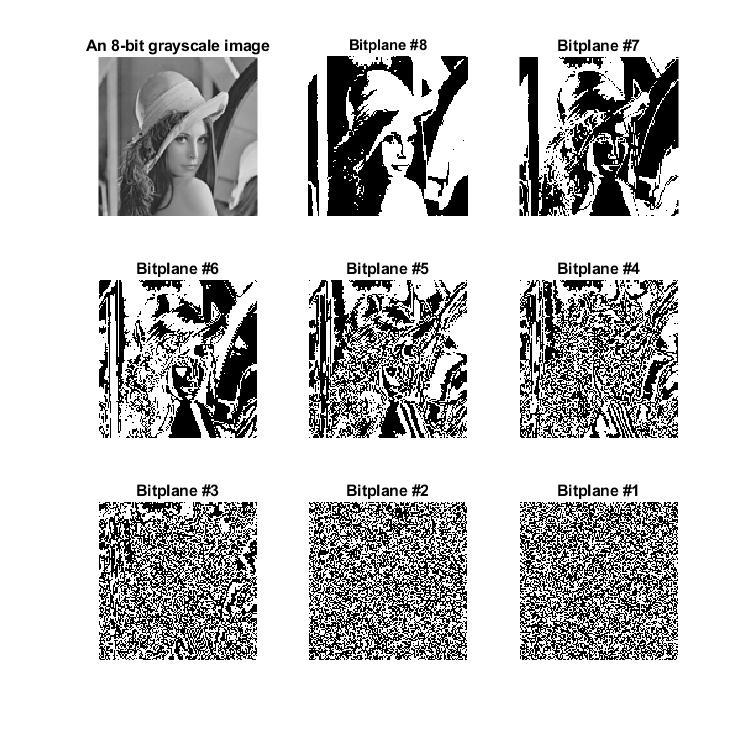
We now introduce the concept of a bitplane in the context of grayscale images. In our 8-bit representation of a grayscale image, there exist 8 bitplanes. To construct the kth bitplane, pull the kth binary digit from each pixel into a new matrix. An example of constructing the 8th bitplane from our matrix is shown below:

You may think of a bitplane as a Logical array – these were introduced in Module 4. Recall that when we visualize a logical array, we view 0 as the color black and 1 as the color white. A small drawing of our bitplane is shown to the right of the binary representation of the bitplane.

**Question 2:** Using the matrix defined above, extract the values for the 6th, 3rd, and 1st bitplanes. You DO NOT need to draw them like we had done for , but please provide the binary value for each element of the matrix.

|  |  |  |
| --- | --- | --- |
|  |  |  |

We now extend the concept of a bitplane to a much larger image. In the subplots shown below (which you will generate during lab; you DO NOT need to make this plot now), we provide the original image along with each of its 8 bitplanes.



**Question 3:** Based on the above image, which bitplanes are most important in describing the features of the original image? Which bitplanes are least important in describing the features of the original image?

Bitplanes 8 and 7 specify the greatest detail within the picture, whereas all the others after that become too convoluted to visualize the picture in any sense.

As you saw in Module 5, Labs 1 and 2, we use the command imshow(img) to display a color or grayscale image. However, to display a bitplane, we must call:

>> imshow(bitplane, ‘DisplayRange’, [0 1]).

For the following questions, you may wish to refer to the documentation of the imshow command.

**Question 4:** What range of values can the pixels of an 8-bit grayscale image take?

0-255

**Question 5:** Why do we need to use this ‘DisplayRange’ of [0 1] when visualizing bitplanes? What would happen if we did not include the ‘DisplayRange, [0 1]’ arguments, i.e. what would happen if we tried to visualize a bitplane as if it were a grayscale image by calling ONLY the following: >> imshow(bitplane)

It is necessary because it specifies only the black and white values of the photo, and narrows it down to only 2 values that the photo should display, versus a range from 0 to 255 which is too broad to use in analysis. If you only called ‘bitplane’, I assume it might show the same photo in grayscale because any further specificity was not established.

*Hint: Think about the range of values a bitplane is allowed to take* vs *the range of values an 8-bit grayscale image is allowed to take.*

As shorthand, the command imshow(bitplane, [0 1]) does the same thing as the command above but saves you time since you do not need to type an additional argument.

***MATLAB Commands Reference***:

MATLAB provides us with a helpful command to extract a bitplane. The syntax is shown below:

>> C = bitget(A,N);

* Input A is an image (grayscale).
* Input N is a number, representing which bitplane you wish to extract.
* Output C is a bitplane, which has the same size as A.
* Summary: The above line of code takes the Nth bitplane from image A and stores it into bitplane C.

*NOTE*: In this context, we assumed A and C were two-dimensional. However, you may also simply extract the Nth bit from a single number (think: single pixel).

MATLAB also allows us to replace the bitplane of one image with the bitplane from another image. The following are some notes about the function bitset:

>> img\_composite = bitset(img\_host, N, bitplane\_embed)

* Input bitplane\_embed is a bitplane that we want to insert into our host image.
* Input N is a number, representing which bitplane in imgHost you wish to replace by bitplane\_embed. number that we want to embed into imgHost.
* Input imgHost represents our host (grayscale) image and is unmodified throughout the code – instead, we make all changes into our new image, img\_composite.
* Output img\_composite is a grayscale image, which is the same size as img\_host.
* Summary: The above line of code lets us replace the Nth bitplane of img\_host with bitplane\_embed and stores the resulting new image into img\_composite.

*NOTE*: The same note from bitget applies here – you can perform bitset on a single number (think: single pixel). Additionally, you may wish to overwrite img\_host by calling

>> img\_host = bitget(img\_host, N, bitplane\_embed)