

## Supplement to Chapter 3 of *The Science of Digital Media* – Digital Image Processing

### [Programming Assignment – Digital Imaging > Dithering](#)<sup>1</sup>

---

#### **Introduction:**

Situations can arise where a certain color is needed for a digital image but the color is not available. It may be that the image is limited to a palette of colors that does not include the desired color. Or it may be that the computer monitor or web browser on which the image is to be displayed cannot represent the desired color. In either case, dithering can be used to simulate a color that is not in an image's or a display device's palette with colors that are.

Dithering works by using colors that are similar to the desired color, arranging them in patterns that are combined by the eye. Because the pixels are small and close together, a good approximation of the desired color can be achieved.

You need to know what dithering is if you decide to change a digital image from RGB mode to indexed color. Your motivation for changing color modes might be so that you can reduce the file size. RGB color uses eight bits for each of the three color channels, resulting in 24 bits per pixel and making possible  $2^{24}$  different colors in the palette. That's over 16 million colors. Digital images in RGB color can be quite large because of the number of bits needed for each pixel. Indexed color uses only eight bits per pixel and allows  $2^8=256$  different colors. Changing a digital image from RGB to indexed color makes the file smaller, but it results in your not being able to represent some of the colors in the original picture.

A simple way to understand the technique of dithering is to try it on a grayscale image. You've probably seen black and white pictures that were dithered to create a grayscale effect. Dithered pictures used to appear frequently in newspapers. In digital imaging, dithering a grayscale image involves changing the bit-depth of the image from eight bits per pixel to one bit per pixel. This means that instead of being able to represent  $2^8=256$  shades of gray, you will be able to show only black or white for each pixel.

The most common dithering algorithms are noise, pattern, and error diffusion dithering. When you change a digital image from RGB to indexed color in a photographic processing program, you may see the noise, pattern, and error diffusion dithering options. Implementing these algorithms will give you a

---

<sup>1</sup> This material is based on work supported by the National Science Foundation under Grant No. DUE-0127280. This worksheet was written by Dr. Jennifer Burg (burg@wfu.edu).

better understanding of how they work and when one is preferable to another.

If you had to change a grayscale image to black and white without dithering, you'd probably take each pixel and, if its value is greater than 127, you'd make it a 1 for white. Otherwise, you'd make it a 0 for black. That would change your picture so that it had sharply divided areas of black and white. This is called *thresholding*.

Noise dithering inserts random noise into the image so that the areas of black and white are not so sharply divided. For each pixel in the image, a random number is generated. If the pixel value is larger than the random number, the pixel value is changed to 1 for white. If it is smaller, the pixel value is changed to 0 for black. Otherwise, a new random number is generated.

Pattern dither uses a pattern called a *mask* or a *halftone screen* to dither the image, making it look less "snowy" than noise dithering. A pattern that has been shown to give good results is

8	3	4
6	1	2
7	5	9

Another good one to try is

1	7	4
5	8	3
6	2	9

The pattern is used in the following manner. First, scale the values in the image so that they are between 0 and 9. You can do this by dividing by 25.6 and dropping the remainder. Then, starting in the upper left hand corner of the image, the 3 x 3 mask is placed over the top leftmost 3 x 3 block of pixels. In each pixel position, if the normalized pixel value is greater than or equal to the mask value, then this pixel is given a value of 1 in the dithered image. Otherwise it is a 0. The mask is then moved over by three pixels. When a row is finished, the mask is moved down by three rows and back over to the left.

In error diffusion dithering, the difference between a pixel's grayscale value and the closest white or black pixel – i.e., the pixel's "error" – is spread among neighboring pixels. The way to do this is simple. For each pixel  $p$ , the error is distributed in a manner reflected in the pattern below.

	$p$	7
3	5	1

Notice that the numbers 7, 3, 5, and 1 add up to 16. The pattern symbolizes that the pixel to the right of  $p$  gets  $7/16$  of the error, the pixel below  $p$  gets  $5/16$  of the error, the pixel below and to the left gets  $3/16$  of the error, and the pixel below and to the right gets  $1/16$  of the error.

The way to implement this is to move from left to right across the pixel rows. For each pixel  $p$ , if the pixel's value is greater than 127, then it is closer to 255 than to 0, so the difference between the pixel value and 255 should be computed. (This will be a negative number). Otherwise, the pixel is closer to 0, so its error is the pixel's own value. The error is distributed among neighboring pixels, as described above. After the error has been distributed over the whole image, the pixels are processed a second time. This time for each pixel, if the pixel value is less than or equal to 127, it is changed to a 0 in the dithered image. Otherwise it is changed to a 1.

Pixels can be processed either left to right across each row, or in an alternating motion from left to right and right to left, weaving back and forth. Note that if the error diffusion weaves back and forth, the order of the error multipliers must be flipped accordingly.

See the interactive demo on dithering for more details.

## Instructions:

### The Assignment

Using the programming language of your choice, implement the three dithering algorithms. Put all three algorithms in one file. Prompt the user for the input image file name, the output file name, the dithering algorithm to use, and in the case of pattern dithering, the pattern file name. Allow the user to dithering more than one image by putting the prompts in a loop.

### Before Writing the Program

To run your program, you will need some raw grayscale image files as input, so create these first or ask your instructor if he or she plans to create them for you. The image files should have one byte per pixel and a header of two integers giving the width and height of the image. The input image files can be created by saving an image (BMP, JPG, etc.) in raw format in a standard image processing program.

### Tip for Creating Input Images



If you can't figure out how to put the width and height on the raw image file as a header, save the image file in .raw format with no header, but make a note of the width and height as you do so. Then write a simple program that reads in the raw file and creates a new file by writing the width and height out first and then the remainder of the image data. Having the width and height on the file as a header saves the user the trouble of typing the width and height in each time an image is dithered.

### Two Possibilities for Implementation – Simulation or the Real Thing

If you can't do bit-level operations in your chosen programming language, then you'll need to simulate the dithering algorithm. That is, you won't actually reduce the eight-bit pixel values to one-bit values and thus reduce the file size. Instead, you'll reduce the eight-bit values that range between 0 and 255 to eight-bit values that are either exactly 0 (black) or 255 (white). (Equivalently, your values could be the eight-bit ascii codes for either the character '0' or the character '1'.) You'll still be able to see the effects of dithering, but your image files won't be any smaller.

If you are able to do bit-level operations, you can do a more realistic dithering where the images change in both appearance and size. You'll need to do some "bit packing" operations to accomplish this.

### Looking at the Dithered Image Files

You don't need to display your image files from within your dithering program. Instead, write the dithered image data to an external file and look at the dithered images after exiting your program using something like Photoshop or MATLAB.

### Tips for C/C++ Programmers



You can use an array of *unsigned chars* for your pixel data. An eight-bit *unsigned char* is just the right size to store a grayscale pixel value. You can take advantage of the fact that *char* and *int* data types are handled interchangeably to some extent in C/C++. That is, you can compare an *unsigned char* data type to the integer constant 127 – essentially treating the *unsigned char* as an integer pixel value.



You'll have to dynamically-allocate memory for each image because image sizes aren't known until run time.



Instead of using the `>>` file-input operator to read your pixel data, use the *read* function associated with random access files. The `>>` operator skips non-printable characters, which will cause you problems if one of your pixel values happens to be the ascii code for a non-printable character. You can use the *read* function to read exactly one byte at a time from the original image file. Then write the bytes back out with the *write* function. (If you're simulating the dithering by writing 0's and 1's as characters, you can use the `<<` operator to write these characters back out to the dithered image file, but be sure to put spaces between them if your file is going to be read by a program such as MATLAB.)



For the encoding process, bit packing can be done by creating an *unsigned long* integer. Assuming that an *unsigned long* is 32 bits, you can shift 0 and 1 bits into this variable using `<<` and `>>` bitshift operators. (The length of an *unsigned long* is system dependent, but you can use the `8*sizeof(unsigned long)` to determine what it is.) Consider using a "mask" value that is "anded" (using `&`) or "ored" (using `|`) with other values to get the 0's and 1's where you want them initially. When the *unsigned long* variable is "full," output it to the external file and initialize the variable to be filled again. Continue until the entire image file is encoded.

### **Ideas for Further Experimentation and Analysis**

- Compare your file sizes before and after dithering.
- If your noise dithering program inserts too much noise and makes your picture look too spotty and indistinct, think of a way to insert less noise and get a better effect.
- Try different masks with your pattern dithering program and analyze the results on different types of images.
- Implement error diffusion dithering two ways – by going left to right across the pixels, and then by weaving back and forth. Compare the results on different types of images.
- Compare the dithering algorithms on different types of images and try to arrive at some conclusions about which algorithm works best for which purpose.
- Implement dithering on RGB color images.