Questions may continue on the back. Please write clearly. What I cannot read, I will not grade. Typed homework is preferable. A good compromise is to type the words and write the math by hand.

The purpose of this exercise is twofold: First, it lets you experiment with a useful technique of image processing, called median filtering. Second, it shows you how to call C code from MATLAB. This is convenient if you want to use the plotting and display facilities of MATLAB, but otherwise you prefer to program in C. In some circumstances, C code can be more efficient than MATLAB code\(^1\), and in those cases calling C from MATLAB may be useful as well.

Median

The median of a set \( S \) of \( n \) numbers is the number \( m \) out of \( S \) that would be in position \( \text{floor}(n/2) \) if the elements of \( S \) were sorted in increasing order. For instance, the median of \( S = \{13, 5, 20, 2, 8, 18\} \) is \( m = 8 \), in position 3 in the sorted list.

One way to compute the median of a set \( S \) is to apply the definition literally: sort \( S \), and take the value in position \( \text{floor}(n/2) \). However, this is needlessly expensive, as we will see shortly.

Median Filtering

Medians are useful in cleaning certain types of noise from images. For instance, the image noisy.jpg on the class web page is corrupted by what is called “salt and pepper” noise, for reasons that ought to be obvious if you look at the image. This noise can be reduced by smoothing the image, because the noise pixels are then averaged with neighboring ones. However, smoothing also blurs edges, and this is often undesirable.

Median filtering reduces salt and pepper noise, and yet harms edges relatively little. It works as follows. An empty output image is created initially, of the same size as the input, but initialized with zeros. Then, each pixel \((r, c)\) of the output image is set to the median of the pixel values in a small \( w \times w \) window around pixel \((r, c)\) in the input image. Usually, \( w \) is chosen to be an odd integer, so the window is symmetric around \((r, c)\). Of course, there is a boundary of pixels around the image for which the window does not fit within the input image. For those boundary pixels we have many options. For simplicity, we will just leave them equal to zero in this assignment.

In the following questions, you will discover why median filtering works, when it fails, and how fast median filtering can be performed.

Figure 1 A “grain of salt” on a dark background next to a light foreground. The “grain” is one pixel in size.

1. Figure 1 shows a ten by ten pixel detail of a bigger picture. Grid lines have been superimposed to make the pixel positions more obvious. Assuming that rows and columns are numbered 1 through 10, the image contains a single “grain of salt” (noise) at pixel \((4, 4)\) next to a boundary between two regions. Let us say that the pixels in the dark region have a value of 20, those in the light region a value of 100, and the grain of salt a value of 200.

(a) What is the value of a median-filtered version of this image at output pixel position \((4, 5)\) and with window size \( w = 5? \)

Just state the value.

(b) Same for pixel at position \((4, 6)\).

\(^1\)This happens less often than you might think, if you write good MATLAB code.
(c) Same for pixel at position (4, 4).

(d) What happened to the “grain of salt” as a result of median filtering?

(e) And to the boundary between dark and light?

(f) How can you modify the shape of the boundary between dark and light pixels in the image sample in Figure 1 so that median filtering with \( w = 5 \) will have a different effect on the boundary itself than what you saw above? Leave the “grain of salt” where it is. [Note: there are many options. Just state one, clearly.]

2. The file `copy.c` available on the class web page contains C code that can be called from MATLAB. As written, it does not do much: it merely copies the input image to the output. In this problem, you will turn this code into a median filter.

First, make sure that the code compiles properly. In MATLAB, type `mex -setup`. MATLAB will ask you a few questions to find out where your C compiler is. Answer \( y \) to the first question, “Would you like mex to locate installed compilers \([y]/n?]\), and then choose the C compiler of your choice from the options you are given (these depend on what is installed on your computer). Confirm your choice when asked.

Now compile `copy.c` from within MATLAB. To this end, make sure that the file `copy.c` is in MATLAB’s current directory. Then type `mex copy.c` to the MATLAB prompt. This will create a new file with base name `copy`, and with an extension that depends on what operating system you are running. In Windows, the file is `copy.dll`. In Unix, the extension depends on the particular flavor of Unix. For instance, with Solaris the file is called `copy.mexsol`. Please look at the External Interfaces/API section of the MATLAB help if you want more details.

Fortunately, you don’t even need to know the name of the compiled file. To run the code from MATLAB, you need an image. Download `noisy.jpg` from the class web page, read it (`imread`) into a MATLAB variable `img`, and convert it to double (`img = double(img);`). You can now use `copy` from MATLAB. Try to type \( c = \text{copy}(\text{img}, 2) \); to the MATLAB prompt. The second argument is useless, but the function `copy` is set up to work for median filtering, which expects half of the window size \( w \) as a second argument (more on this later). You may want to check that `copy` did what you think it should do. Easy, no?

Let us now look inside `copy.c`. There are three functions:

- **kth_smallest** computes the \( k \)-th smallest of \( n \) numbers passed in the vector \( a \), using an efficient algorithm developed by Niklaus Wirth. A macro `median` shows you how to call `kth_smallest` so as to obtain the median.

- **imgMedian** has a misleading name. All it does now is to copy one array to another with the standard-library function `memcpy`. Do not throw this function away. It may turn out to be handy for your work. This is the function you will change so that it does what its name implies. The argument list is fine as is.

- **mexFunction** is the interface between your code and MATLAB. You are not supposed to change the name of the function, or its argument list, because MATLAB expects to find this exact header in any file you want to compile with `mex`. The body of `mexFunction` is set up to work properly for median filtering. In the future, it ought to be easy for you to modify this function to write other MATLAB-callable functions, if you need to do so. For this assignment, leave `mexFunction` alone.

(a) Make a copy of `copy.c`, and rename it to `immedian.c`. Change the body of the function `imgMedian` in `immedian.c` so that it does median filtering on image \( \text{in} \) and with window size \( w = 2 \times \text{hwin} + 1 \) (this guarantees that \( w \) is an odd integer). The arguments \( m \) and \( n \) are the number of rows and columns in the image, and \( \text{in} \) is a vector of doubles that contains the pixel values one column at a time (first column followed by second column, and so forth). Values in \( \text{out} \) are expected to be written in the same order. Leave margins of the picture black (zeros). Storage for the output image \( \text{out} \) is already allocated by `mexFunction`. Hand in your code for `imgMedian` (a printout is fine). Warning: `kth_smallest` overwrites its input \( a \).

(b) Run your code on `noisy.jpg` as follows:

\[
\text{tic; out = immedian(img, 1); time = toc}
\]

and state the value of `time`. Please state what machine and operating system you are running your code on. If you know, also state what CPU you are using (including the clock speed), and how much memory it has. CPU and memory information is not required for full credit.
(c) Display the output image out (use imagesc and colormap gray) and hand in a printout (use print to print).

3. The code in copy.c that computes the median of a vector of numbers assumes that each median computation is entirely new. However, in image median filtering, most of the pixels in the window for pixel \( (r, c+1) \) were already present in the window for pixel \( (r, c) \). In other words, two consecutive median computations are usually performed on overlapping sets of pixels. This exercise tries to exploit this redundancy for greater efficiency. This makes sense only when the window size \( w \) is very large. For smaller windows, it is hard to beat the algorithm we used previously.

In the questions that follow, let \( p_{r,c}(k) \) for \( k = 0, \ldots, 255 \) be the histogram of the values in the current window (of given size \( w \)). This is defined as the number of pixels in the window around pixel \( (r, c) \) that have value \( k \) (we only consider black-and-white images with 256 levels of gray).

No implementation is required in this exercise.

(a) Explain how to compute the median \( m \) in a window around pixel \( (r, c) \) from the histogram \( p_{r,c}(k) \) (for fixed \( r \) and \( c \), the latter is a vector with 256 values). Your explanation can be in the form of English sentences, a formula, pseudo-code, or code.

(b) Let \( p_{r,c} \) be the vector that collects the 256 values of \( p_{r,c}(k) \) for \( k = 0, \ldots, 255 \). Find an efficient way to compute the vector \( p_{r,c+1} \) from \( p_{r,c} \). The window size, \( w \), is a given, odd integer. See the previous question for acceptable answer formats.

(c) Assume that a similar method exists for changing \( p_{r,c} \) to \( p_{r,c+1} \). Then, median filtering can traverse the image in some suitable order, run the appropriate method for updating the vector \( p_{r,c} \) at each pixel \( (r, c) \), and compute the median from the histogram. In what order should the input image be traversed in order to save as much as possible on the necessary amount of time and storage used?

Project Topics

*This and the following homework assignments suggest topics for the final project for this class. Just pick one of them, do it, and you are done with the project requirement. Official project deadlines are set in the last three weeks of the semester. However, you can complete and hand in your project earlier if you like. Sometimes, more projects are suggested. For instance, this time there are two suggestions. You only need to do one project over the whole semester.*

A completed project always includes a report that explains what you do and shows results. Suggestions on the report format are sometimes included.

1. Write C code, callable from MATLAB, that median-filters any input image with windows of any odd size.

   For smaller windows, the algorithm used in \texttt{immedian.c} is most efficient. For bigger windows, redundancy should be exploited as illustrated in the last problem above. It is your responsibility to determine when to switch from one method to the other. One way to find out is to estimate the number of operations as a function of \( w \), and determine the break-even point theoretically. Alternatively, or, better, in addition, an empirical evaluation can be used (use the MATLAB tic and toc functions to time your program on large images, or on many images). It is useful to compare the performance of your code with that of the MATLAB image toolbox function \texttt{medfilt2}.

   In addition, your code should do something better at image boundaries than just leaving them black. For instance, you could shrink the window size as you approach the boundaries. If you only shrink as much as strictly necessary, your windows become rectangular, so you may want to keep that in mind when you start writing your code.

   Your report would include an explanation of your code and of your calculation of the break-even point, a diagram that shows your experimental results for the break-even point, examples of inputs, outputs, and running times on one or more images with different window sizes, and a plot of running times as a function of window size for a few different image sizes. If you use different algorithms for computing medians, list the appropriate bibliographic references.

2. The paper “Bilateral Filtering for Gray and Color Images” by C. Tomasi and R. Manduchi (Proceedings of the Sixth International Conference on Computer Vision, Bombay, India, pp. 839–846, January 1998) discusses a method for smoothing images while preserving edges. The paper is available through the class web page. Read the paper, and write an efficient function callable from MATLAB that implements the method.

   There are no immediately obvious ways to exploit redundancy in this method, so efficiency would be achieved essentially by good programming (presumably in C) and perhaps a judicious use of table lookups.
A literal implementation, although simple, would be slow, so there is ample room for improvement here. Your implementation would fill a real, practical need, since the bilateral filter has encountered quite a bit of interest, and I do not know of a very efficient implementation.

The format of your project report would be similar to that for the previous project, with an additional section that explains the bilateral filter in your own terms. Also, the effects of filter parameters on the result are less understood than for the median filter, so your input/output examples would come with a more detailed discussion of strengths and drawbacks of the filter.