COMP 116
Programming Assignment #4
Working with binary images
Due Sunday, October 17th by 11:59pm

Assignment resources
In this assignment, we will be using binary images from the Laboratory for Engineering Man/ Machine Systems at Brown University <*http://www.lems.brown.edu/~dmc/*>. You can use any of the binary images on that page to test your program and to generate your final published result. NOTE: Record the web address of the image or images you use in your published program; you will need to report this information as part of the assignment.

Binary images
Binary images are those images in which each pixel is either black (0) or white (1). This is different from, for example, a black-and-white television or a grayscale image, where each pixel can be one of many shades of gray. An example can be seen in Figure 1.

Figure 1: Binary image of a walking man

Silhouettes are often binary images. See the results of a Google image search for "silhouette": <http://www.google.com/images?q=silhouette>.

In this assignment, you will write code that allows you to perform some simple image processing techniques on binary images (similar to techniques that are used in Adobe Photoshop or other image processing programs), including locating edges and eroding or dilating the image. More details on these tasks are below.

How to load and display binary images:
If you call `imread(imageName)`, the resulting matrix will be full of integers (0s and 1s). If you try to display the image directly, you will not see a white object on a black background. That is because MATLAB, by default, will treat these values as grayscale values between 0 and 256, and so the entire image will be black or very close to black.

The easiest way to address this problem is by creating a matrix of logicals instead of a matrix of integers. You can do this with the `logical(matrixName)` command as follows.

```
img = imread(imageName);
binaryImg = logical(img);
```

Once you have your image stored as a matrix of logicals, `image(binaryImg);` will display your image properly.

Assignment structure

In this assignment, we will be using multiple M-files for the first time. You will have one M-file, for example Assignment4.m, that calls functions defined in other M-files in order to perform the tasks required in the assignment. Specifically, you will have to define the seven functions listed below.

For illustration, the results of applying each function to the test image shown at right (of a star) are shown along with each function description.

Scrolling functions:

These functions enable the scrolling of an image in the four cardinal directions. All scrolled images should be *the same size* as the input image, with the contents shifted. To simplify these functions, you can make the following assumptions.

(1) No invalid arguments will be passed (*i.e.* `amount` will always be "reasonable"). So you do not have to check the validity of `amount` or `image`.

(2) Whenever you scroll an image, the empty space left behind should be left black. (That is, when part of the image scrolls off one side, you should create an equal number of 0 pixels on the other side so that the image stays the same size.

• `scrollLeft(image, amount)`

`scrollLeft` takes 2 parameters: a binary image, `image`, and an integer, `amount`, that is the distance *in pixels* that the image should scroll. It produces a new image, which is `image` scrolled to the left by `amount` pixels.

• `scrollRight(image, amount)`

`scrollRight` takes 2 parameters: a binary image, `image`, and an integer, `amount`, that is the distance *in pixels* that the image should scroll. It produces a new image, which is `image` scrolled to the right by `amount` pixels.

• `scrollUp(image, amount)`

`scrollUp` takes 2 parameters: a binary image, `image`, and an integer, `amount`, that is the distance *in pixels* that the image should scroll. It produces a new image, which is `image` scrolled up by `amount` pixels.

• `scrollDown(image, amount)`

`scrollDown` takes 2 parameters: a binary image, `image`, and an integer, `amount`, that is the distance *in pixels* that the image should scroll. It produces a new image, which is `image` scrolled down by `amount` pixels.

Image processing functions:

As above, the output images should always be the same size as the input image, and you do not have to check for argument validity.

HINT: These functions are easy to write if they call the scrolling functions defined above.

• `findBoundary(image)`

`findBoundary` takes 1 parameter: a binary image, `image`. It returns a new image where the only white pixels are the boundary pixels in `image`.

• `erodeImage(image)`

`findBoundary` takes 1 parameter: a binary image, `image`. It returns a new image that is an *eroded* version of `image`.

• `dilateImage(image)`

`findBoundary` takes 1 parameter: a binary image, `image`. It returns a new image that is a *dilated* version of `image`.

Stubs for the scrolling functions have been provided for you. A stub is a what we call a function that has the same name and parameters as the desired function, and will perform some minimum functionality that allows you to test the rest of your program, but does *not* do the task it is expected to. In this case, the stub functions return an all-black image. It is up to you to write the code to make the desired result images.

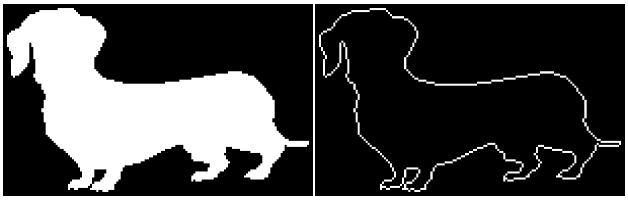
Stubs have *not* been provided for the image processing functions. You will need to create new M-files for each of these. You can use the scrolling functions as models.

Assignment hints:

Although we've seen `if` and `else`, they are not needed for this assignment. You can use them if you feel the need, but the entire assignment can be done just by using two features of MATLAB: slicing and logical operators (which, you may remember, can be applied to entire matrices at once). **DO NOT USE LOOPS**. Solutions that use loops will not be given full credit.

Boundary detection

A common image-processing task is to identify the boundary (outline) of an object. Conceptually the boundary is the set of all pixels in the foreground that are adjacent to the background. Here is an image of a dachshund and another image of its boundary.



Hints about boundary detection:

We can use the scrolling functions from the first part of this assignment to determine if a pixel is on the object boundary or not. Suppose that in our MATLAB workspace, *I* contains the image data, and *LI* is the result of the command:

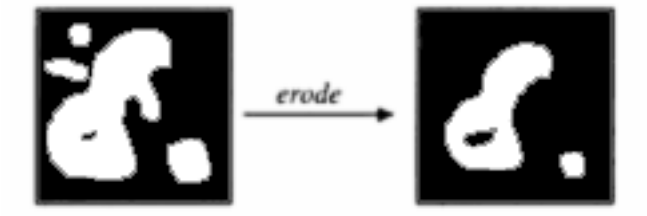
LI = scrollLeft(I, 1);

A pixel, (*i, j)* is on the ***right*** edge of the boundary if and only if *I(i, j)* is foreground and *LI(i, j)* is background. You can use the other scrolling functions from part 1 to identify the pixels on the other sides of the boundary. Combine these tests together with the correct *logical operator*, and you will be able to identify the entire boundary of the object.

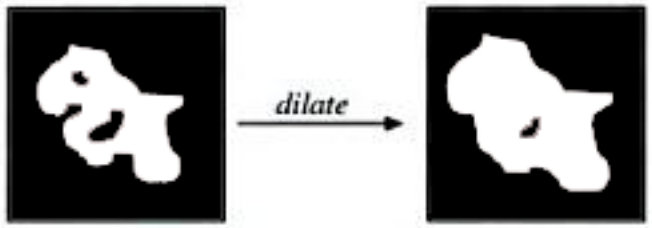
Erosion and dilation

Erosion and dilation are two common image processing operations. They are essentially opposites, although technically they are not *inverse* functions of one another. (After you read the rest, can you see why?)

Erosion shrinks the size of the foreground objects ("foreground" = white or 1-valued pixels). Alternatively, it expands the background ("background" = black or 0-valued pixels). As a result, it smoothes the boundary of an object by removing features that are "too small": peninsulas, bridges, or very small objects. An example of erosion can be seen in the following figure (taken from *Handbook of Image and Video Processing*, by Alan Conrad Bovik).



In contrast, *dilation* expands the size of the foreground objects ("foreground" = white or 1-valued pixels). Alternatively, it shrinks the background ("background" = black or 0-valued pixels). As a result, it smoothes the boundary of an object by "filling in gaps": small holes or inlets An example of erosion can be seen in the following figure (taken from *Handbook of Image and Video Processing*, by Alan Conrad Bovik).



If you are interested, the pages of Bovik's book relating to erosion and dilation are on Google Books, <http://books.google.com/books?id=P7y2qV0qys8C&lpg=PA49&ots=F3drKgZ3I1&dq=eroding%20and%20dilating%20images&pg=PA47#v=onepage&q&f=false>.
The material presented in the book is somewhat advanced; you DO NOT need to read it in order to complete the assignment, and it WILL NOT appear on tests or quiz. It is purely for your own interest if you'd like to learn more.

Hints about erosion/dilation:
It is easy to erode an image – just change the intensity of the boundary from 1 to 0. To dilate an image, you will need to write a new logical test to find pixels that are in the background of the original image, but are adjacent to a foreground pixel. You can use the scrolling functions from above to identify these pixels.

How to demonstrate your program

To demonstrate that your functions are working properly, please write a single script file, Assignment4.m, that performs the following tasks (note that while all this happens in a single file, Assignment4.m, this script WILL be calling functions that are defined in other M-files) :

(1) Display the binary test image you are using, unmodified.
(2) Scroll that test image up by 10 pixels and display the result
(3) Scroll the test image down by 10 pixels and display the result.
(4) Scroll the test image left by *half its width* and display the result.
(5) Scroll the result image from (4) back to the right by the same amount and display the result.
(6) In a comment, answer this question in your own words. Is the image from (5) the same as the original image? Why or why not?
(7) Compute and display the boundary image.

(8) Dilate your test image. Then dilate that result. Then dilate *that* result. (So the original image will now have been dilated three times). Display the result.
(8) Erode your test image. Then erode that result. Then erode *that* result. (So the original image will now have been eroded three times). Display the result.
(9) Take the result image from (8), and dilate it 3 times. Display the result.
(10) In a comment, answer this question in your own words. Is the image from (9) the same as the original image? Why or why not?
(11) Find the boundary of the test image. Dilate it once. Display the result.
(12) Dilate the test image once, then find the boundary of the result. Display this image.
(13) In a comment, answer this question in your own words. What is the difference, if any, between the images from (11) and (12)?

Grading notes

If you do not include a discussion with your thoughts about the assignment, what you had problems with, or what you learned in this assignment, *etc.*, **you will automatically lose 25% of your grade.**

If you do not submit all the result images and discussion in a single published file (either HTML or PDF is fine), **you will automatically lose 50% of your grade.**

If your images are not visible in the published file (whether you neglected to copy them along with your HTML file, or you closed the figure window before the image rendered, or an error in your code prevented the image from displaying, or whatever else), **you will not receive credit for any component where images are missing.**

If you are unclear whether your file satisfies these requirements, contact one of the TAs before the due date and we will check it for you.

As with the previous assignments, publish your completed script file (by typing `publish Assignment4.m;` (which generates an HTML file and images) or `publish(Assignment4.m, 'pdf');` (which generates a PDF) in the MATLAB command window. Create a folder inside your shared Dropbox folder named Assignment4. Copy the published html directory or pdf file into that Dropbox folder. **Also copy the M-files containing your functions into that folder**.

Required Components: [100 pts]
The (X)-es refer to the steps in the "How to demonstrate your program" section.

(Note that all code for each of these components should be included in your published script.)
1. Display your original image and state its (a) web address and (b) dimensions. [5 pts]
2.Display the scrolled image from (2). [5 pts]
3. Display the scrolled image from (3). [5 pts]
4. Display the scrolled image from (4). [10 pts]
5. Display the scrolled image from (5). [10 pts]
6. Answer the question from (6). [5 pts]
7. Display the boundary image from (7). [10 pts]
8. Display the dilated image from (8). [10 pts]
9. Display the eroded image from (9) [10 pts]
10. Answer the question from (10). [5 pts]
11. Display the image from (11). [10 pts]
12. Display the image from (12). [10 pts]
13. Answer the question from (13). [5 pts]