# 3	ECE 172a	Pamela Cosman	1/12/12

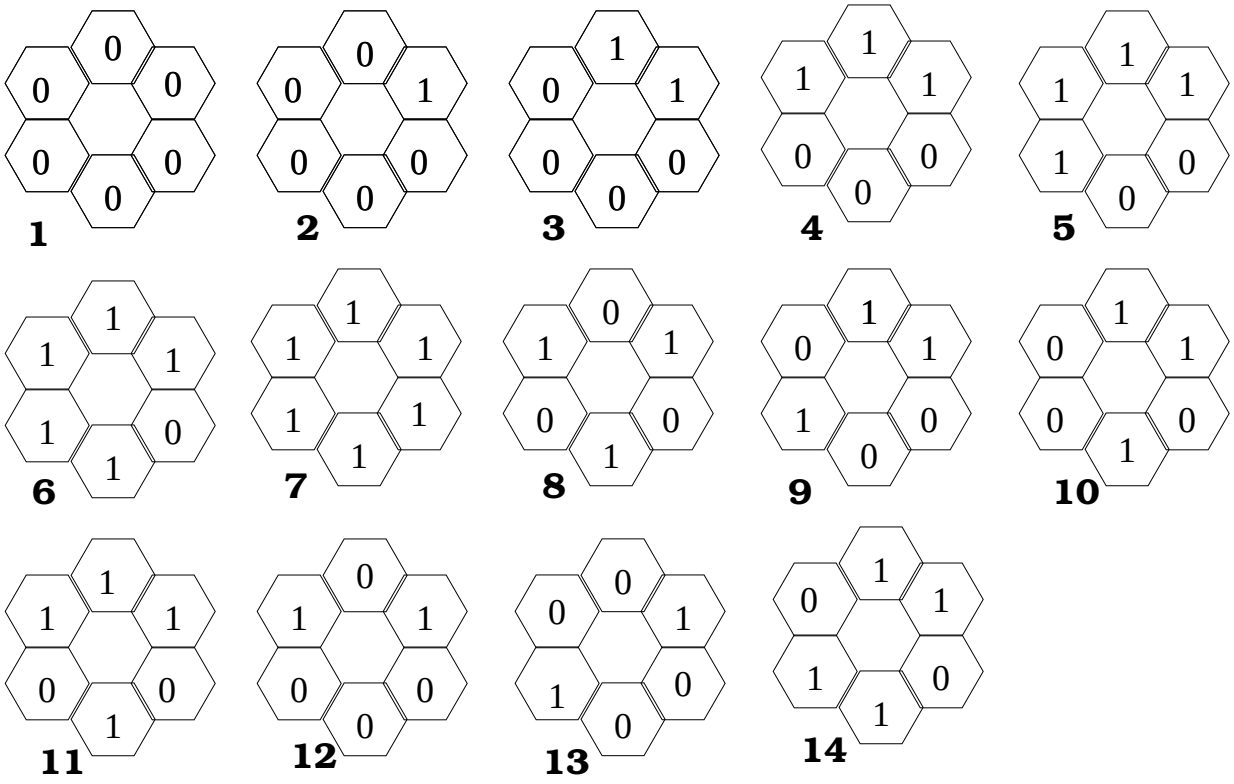
HOMEWORK 1

Due Monday January 23, 11am (start of class)

For all the problems involving Matlab, please turn in your edited sequence of Matlab commands, and the text of any functions that you write, as well as the descriptive prose of your results.

1. **Morphological operators on a hexagonal lattice:**

   If we ignore rotations, then there are 14 possible surrounds of a cell on a hexagonal lattice, as follows:



   We can uniquely define the operation of an iterative modification scheme by specifying:

   - The set A of surrounds (for which $a_{ij} = 1$)
   - The Boolean function $L(a, b)$ (where $c_{ij} = L(a_{ij}, b_{ij})$)
   - The number of iterations, n
   - The number of subfields, f, into which the image tesselation is divided

We use ab to denote *a and b*, $a + b$ to denote *a or b*, and $\bar{a}$ to denote *not a*.

Consider the following iterative modification schemes:

(1) $A = \{1\}$ $L(a, b) = b$ $n = 1$ $f = 1$
(2) $A = \{\}$ $L(a, b) = ab$ $n = 1$ $f = 1$
(3) $A = \{1\}$ $L(a, b) = ab$ $n = 1$ $f = 1$
(4) $A = \{2\}$ $L(a, b) = b\bar{a}$ $n = 1$ $f = 1$
(5) $A = \{7\}$ $L(a, b) = ab$ $n = 1$ $f = 1$
(6) $A = \{4\}$ $L(a, b) = ab$ $n = 1$ $f = 1$
(7) $A = \{3, 4\}$ $L(a, b) = \bar{a}b$ $n = \infty$ $f = 3$
(8) $A = \{1, 2, 3\}$ $L(a, b) = \bar{a}b$ $n = \infty$ $f = 1$
(9) $A = \{5, 6, 7\}$ $L(a, b) = a + \bar{a}b$ $n = \infty$ $f = 3$
(10) $A = \{8\}$ $L(a, b) = ab$ $n = 1$ $f = 1$
(11) $A = \{1\}$ $L(a, b) = ab$ $n = 1$ $f = 1$ followed by $A = \{7\}$ $L(a, b) = a\bar{b}$ $n = 1$ $f = 1$
(12) $A = \{7\}$ $L(a, b) = a\bar{b}$ $n = 1$ $f = 1$ followed by $A = \{1\}$ $L(a, b) = ab$ $n = 1$ $f = 1$
(13) $A = \{1, 7\}$ $L(a, b) = ab$ $n = 1$ $f = 3$
(14) $A = \{1\}$ $L(a, b) = ab$ $n = 5$ $f = 1$
(15) $A = \{7\}$ $L(a, b) = ab$ $n = 5$ $f = 1$

Note that when we include a surround in the set A, we also include all rotated versions of that surround. For each of these schemes, find the description in words below that best fits the action it performs. Some descriptions may be used more than once, others not at all. For each answer, include some explanation.

(A) Fills in small cavities
(B) Cleans up (removes) isolated cells that are one
(C) Keeps only isolated cells that are one
(D) Removes edges of blobs, keeping the interior
(E) Removes 5 layers of edge pixels from blobs (for example, any blob whose maximum dimension is 10 pixels or less will be erased)
(F) Removes interiors of blobs, keeping only the edges
(G) Cuts off all appendages (that is, thin lines) and removes tiny blobs
(H) Mark all places where 3 thin lines come together
(I) Resets all cells to zero
(J) Marks isolated pixels of either color
(K) Removes edges of blobs, keeping only the interior, but making sure not to completely erase any blob
(L) Removes ends of lines
(M) Reproduces the input image without change (identity operator)
(N) Skeletonizes until lines are only one picture cell wide
(O) Complements all picture cells in the image
(P) Marks all corners, flushes the rest
(Q) Marks any isolated zero cell

2. Use the command: eye = imread('eyechart.tif'); to read in the binary image `eyechart` which is of size 256 by 256. Write a routine which takes as input arguments a binary image and a percentage p of pixels to be randomly flipped. Flipping a pixel means that if it is white it will be made black, and if it is black it will be made white. This is sometimes called salt-and-pepper noise. (You will find the commands xor and rand useful). Use your routine to generate 3 different versions of the `eyechart`: one with low levels of noise ($p = 1\%$), one with a medium level of noise ($p = 5\%$), and one with lots of noise ($p = 20\%$).

   We would like to clean up these noisy versions. Under the command bwmorph, you will find the operation 'clean' which does isolated pixel removal. (Type help bwmorph). That is, a black pixel entirely surrounded by white pixels gets flipped, as does a white pixel entirely surrounded by black pixels. Apply this to the three noisy versions. How well does it do? Also under bwmorph, you will find the operations open and close. Try cleaning the three noisy versions by doing opening followed by closing. How does this compare to the isolated pixel removal? Are the results different if we do closing first and then opening?

3. Use the command: a = imread('bincell256.tif'); to read in a thresholded binary version of some cells. Use imshow to look at it. We see that the binarization was not very good. Many of the cells in the binary version are fragmented; portions of them were not detected as object pixels. Look at the various commands available under bwmorph. What combination of operators can you use to get the cells as "complete" as possible? That is, we would like the interior portions of the cells to be 1 pixels, but we want the cells to end up the correct size. So, for instance, we can't just do 10 rounds of dilating, since that would certainly declare all the interior pixels to be 1's, but the cells would also end up larger than their true size.

   Try various different combinations of operators, and report your results.

4. The "raggedness" of the boundary of a cell or a cell nucleus provides some information about disease condition, as does the overall shape. A machine vision system might be used to measure cell "raggedness" for a pathologist. See if you can invent a useful way of measuring this. We will need some test images to evaluate our raggedness parameter.

   Generate a small binary square with a perfectly smooth boundary as follows:

```
>> sq = zeros(80);
>> sq(20:60,20:60) = ones(41);
```

   Now generate a version of the square with a ragged edge as follows:

```
>> ragged1 = sq;
>> for i = 20:2:60,
ragged1(20,i) = 0;
ragged1(60,i) = 0;
ragged1(i,20) = 0;
ragged1(i,60) = 0;
end
```

We will generate two more versions where the "fringes" are even longer:

```
>> ragged2 = ragged1;
>> for i = 20:2:60,
ragged2(21,i) = 0;
ragged2(59,i) = 0;
ragged2(i,21) = 0;
ragged2(i,59) = 0;
end
>> ragged3 =ragged2;
>> for i = 20:2:60,
ragged3(22,i) = 0;
ragged3(58,i) = 0;
ragged3(i,22) = 0;
ragged3(i,58) = 0;
end
```

Now, try using some combination of morphological operators on these 4 versions so that you come up with a single number (for each little image) that represents the boundary raggedness for that binary shape. (You will find the bwperim command useful. Note that it can calculate both 4-connected and 8-connected perimeters.)

You would like your calculated parameter to be monotonic (either increasing or decreasing). For example, if the square with a smooth boundary produces a raggedness $\mathcal{R}$ of zero, so we could write

$$\mathcal{R}(SmoothSquare) = 0$$

then we would like to have

$$0 < \mathcal{R}(ragged1) < \mathcal{R}(ragged2) < \mathcal{R}(ragged3)$$

since these have progressively more ragged boundaries. Explain why your parameter is a good way to measure the raggedness of the edge of a binary shape.

5. Boundary raggedness of the real cells: Use the imcrop command to crop out 3 cells from the original cell image. Choose one which seems to be very smooth, one which has a ragged boundary, and one cell which looks completely fragmented. Compute your raggedness parameter $\mathcal{R}$ for each one of the 3. Discuss your results. Are they reasonable?