ECE 178 Hwk #2 Solutions

1) Problem 2.9

(a) The total amount of data (including the start and stop bit) in a 256 gray level or 8bit ,1024 x 1024 image, is $(1024)^2$ x [8 + 2] bits. The total time required to transmit this image over a At 56K baud link is $(1024)^2$ x [8 + 2] /56000 = 187:25 sec or about 3.1 min.

(b) At 750K this time goes down to about 14 sec.

2) Problem 2.10

The width-to-height ratio is 16/9 and the resolution in the vertical direction is 1125 lines (1125 pixels in the vertical direction). It is given that the resolution in the horizontal direction is in the 16/9 proportion, so the resolution in the vertical direction is (1125)x(16/9) = 2000 pixels per line. The system paints a full 1125x2000, 8-bit image every 1/30 sec for each of the red, green, and blue component images. There are 7200 sec in two hours, so the total digital data generated in this time interval is (1125)(2000)(8)(30)(3)(7200) = 1.166 x 10^{13} bits, or 1.458 x 10^{12} bytes (i.e., about 1.5 terrabytes). These numbers show why image data compression is so important.

Note: Many books where worded to say that there were 8 pixels for each color. This does not make any sense and should have said 8 bits of color for each of the three primary colors.

3) Problem 2.11

Let p and q be as shown in Fig. P2.11. Then, (a) S1 and S2 are not 4-connected because q is not in the set $N_4(p)$ (b) S1 and S2 are 8connected because q is in the set $N_8(p)$ (c) S1 and S2 are *m*-connected because (i) q is in $N_D(p)$, and (ii) the set $N_4(p) \cap N_4(q)$ is empty.

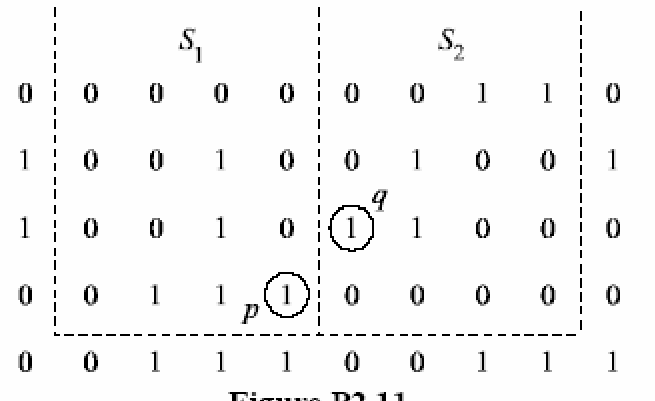

Figure P2.11

4) Problem 2.12

The solution to this problem consists of defining all possible neighborhood shapes to go from a diagonal segment to a corresponding 4-connected segment, as shown in Fig. P2.12. The algorithm then simply looks for the appropriate match every time a diagonal segment is encountered in the boundary.

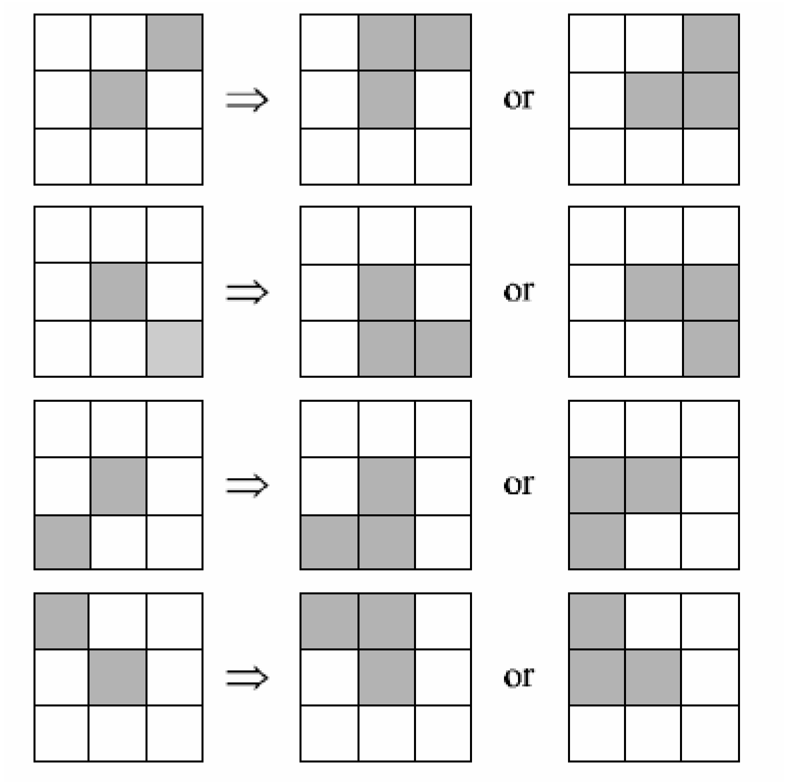

Figure P2.12

5) Problem 2.16

(a) A shortest 4path between a point p with coordinates (x; y) and a point q with coordinates (s; t) is shown in Fig. P2.16, where the assumption is that all points along the path are from V . The length of the segments of the path are |x – s| and |y - t|, respectively. The total path length is |x - s| + |y - t|, which we recognize as the definition of the D4 distance, as given in Eq. (2.516). (Recall that this distance is independent of any paths that may exist between the points.) The D-4 distance obviously is equal to the length of the shortest 4-path when the length of the path is |x – s| + |y – t|. This occurs whenever we can get from p to q by following a path whose elements (1) are from V; and (2) are arranged in such a way that we can traverse the path from p to q by making turns in at most two directions (e.g., right and up). (b) The path may of may not be unique, depending on V and the values of the points along the way.

Note: V is the set of values that your path is allowed to trace along.

6) Problem 2.17

(a) The D-8 distance between p and q (see Fig. P2.16) is defined as max(|x – s| ; |y – t|). Recall that the D-8 distance (unlike the Euclidean distance) counts diagonal segments the same as horizontal and vertical segments, and, as in the case of the D4 distance, is independent of whether or not a path exists between p and q. As in the previous problem, the shortest 8 path is equal to the D8 distance when the path length is max(|x - s| ; |y - t|). This occurs when we can get from p to q by following a path whose elements (1) are from V , and (2) are arranged in such a way that we can traverse the path from p to q by by traveling diagonally in only one direction and, whenever diagonal travel is not possible, by making turns in the horizontal or vertical (but not both) direction. (b) The path may not be unique, depending on V and the values of the points along the way.

7) Problem 2.19

The operator H is linear if H(aS1 + bS2) = aH(S1) + bH(S2) holds for $\forall a,b \in \Re$. The median of a set of numbers is such that half the values in the set are below median value and the other half are above it. In this case H is the median operator. A simple example will suffice to show that linearity condition is violated by the median operator.

Let S1 = {1,-2, 3} S2 = {4, 5, 6} and a = b = 1. H(S_1) = 1, H(S_2) = 5. Therefore, H(S_1) + H(S_2) = 6. On the other hand, S_1 + S_2 is the element-by-corresponding-element sum of S_1 and S_2, and S_1 + S_2 = {5, 3, 9}. Therefore, H(S_1 + S_2) = 5. Since we found a,b, S_1 and S_2 for which linearity condition does not hold, i.e. H(a S_1+b S_2) ≠ aH(S1)+bH(S_2), it follows that the median is a nonlinear operator.

8)

$$a)\begin{bmatrix} 1 & 2 & 2 & -2 & -3 \\ 0 & 4 & \underline{\underline{0}} & -4 & 0 \\ 3 & 2 & -2 & -2 & -1 \end{bmatrix} \text{ b) } \begin{bmatrix} 4 & 13 & \underline{\underline{28}} & 27 & 18 \end{bmatrix} \text{ c) } \begin{bmatrix} 4 & \underline{\underline{13}} & 28 & 27 & 18 \end{bmatrix}$$

9) MATLAB Problem
A few comments about the code below. First, note the convenience in using a function to carry out the halftoning procedure on the image. Second, note the use of multidimensional arrays for storing the halftoning patterns. This allows us to keep the code compact. You may also want to look at the method used to map the gray levels to the halftone: one line of code was enough to implement a basic quantizer.

Also note that in this program the original image is resized to ½ of the original size. Note that the ½ value will change depending on the resolution your printer uses. The idea is that if your printer prints at 150 dots per inch and your paper is 8 inches across then your resulting image better not have a size bigger than the size*print resolution. This leads to the need to resize the image but the resizing must be done before halftoning because if it is done after the lines removed will ruin the halftone pattern that you created.

MATLAB test program

```
% NAME: Hw2_sol.m
% DESC: Test for the haltoning function for the HW #2 programming assignment
% AUTHOR: Marco Zuliani
% DATE: 1-26-2003
close all clear all clc
% read the image
I = imread('image1.jpg');
% test image
% I = zeros(256);
% for ind = 1:256;
% I(ind, :) = ind-1;
% end
% I = uint8(I);
% rescale it in order to have the halftoned image fitting in the screen
Ir = imresize(I, 0.5*size(I), 'bicubic');
% create the halftoned image
Iht = halftone(Ir);
% show results
imshow(Iht); title('Halftoned image')
```

MATLAB halftoning function

```
function Iht = halftone(I)
% Iht = halftone(I)
%
% DESC:
% Returns the halftoned version of the input image I. Halftoned image dimensions
% are three times bigger then those of the original image.
%
% INPUT:
% I = input image
%
% OUTPUT:
% Iht = output image
%
% AUTHOR:
% Marco Zuliani
% Let's prepare the halftones: note the use of multidimensional arrays.
halftones = zeros(3, 3, 10);
% nice way to init the halftones preserving your fingertips...
halftones(1, 2, 2:10) = 255; halftones(3, 3, 3:10) = 255;
halftones(1, 1, 4:10) = 255; halftones(3, 1, 5:10) = 255;
halftones(1, 3, 6:10) = 255; halftones(2, 3, 7:10) = 255;
halftones(3, 2, 8:10) = 255; halftones(2, 1, 9:10) = 255;
halftones(2, 2, 10) = 255;
% quantize the original image. This formula allows us to span between 0 to 10
%(halftones indices)
%Note that I am dividing by 256 since we need to count also the value zero.
Iq = floor(double(I) / 256 * 10) + 1;
% re-map the quantized values using the halftones
% Note that when doing loops it is a good programming policy (read: more
% speed) to initialize the arrays out of the loop.
Iht = zeros(3*size(I)); for i = 1:size(I, 1)
for j = 1:size(I, 2)
ii = 3*i;
jj = 3*j;
Iht(ii:ii+2, jj:jj+2) = halftones(:, :, Iq(i, j));
end
end
% enforce unsigned int
Iht = uint8(Iht);
return
```