CSE107 Lab 1 Report

Binary Image Rendering Using Halftoning

Darren Correa

2/17/2022

Abstract

For this lab, our goal is to learn how to manipulate images using their associated matrix values. For this to happen, we must understand how an image is rendered in a computer. Since computers can't "see" an image, they are composed of numbers inside of a matrix. The matrix size is the dimension of the image. Each pixel is associated with a value to determine the color of that pixel. Our goal is to take those values and associate a binary value to it. The values of each pixel can be between 0 and 255. For this assignment, the approach I took required grouping several pixels together and determine if the group of pixels should be 0 or 255. I believe my results are somewhat correct but I can be missing something to have the correct output image.

Technical Discussion

The halftone function can be broken into 3 parts. First initialization. Since the instructions asked to create an output image, the first part of the function should allocate a matrix the same size as the input image. For this the zeros' function was sufficient to allocate an "empty" matrix of the correct size. The second part is creating the 3x3 blocks within the image. To do this, using two loops (iterating i and j), we treat each value as a pixel, at each matrix value, we take the average of the pixel at the iteration point, along with the pixels surrounding it in a 3x3 area. Line 21 of the halftone functions shows this calculation. For sizing, the modulo function is used to ensure when doing the average calculations, the iterations do not exceed the area of a 3x3 block. If we did not have the boundary in place, when iterating at an edge, there would be an error since there are no more values past the edge of the matrix. After taking the average, we can assign binary values to the 3x3 areas. If the average is less than half of 255 we can assign 0, if not assign 255 to the area. After all iterations are complete, out output image is complete and our scripts are able to generate the images using imshow() and saved with imwrite().

Discussion Results

I am pleased that the output images produced show a version of halftoning. After viewing examples of halftoned images from Google searching I realize, my implementation of the function is somewhat incorrect. I'm assuming my version of changing the values in a 3x3 area is incorrect because blocks are not allocated. In my version each iteration allocated a 3x3 area, but it does not skip directly to an adjacent 3x3 area each time, only to the next column. That was the main challenge I faced over the two weeks to work on the lab. Creating the original wedge image was a sinch. Create a 256x256 matrix of zeros and fill each row with 0 to 255 and then halftone the wedge image. This is when I fully realized something wasn't quite right. Image 8 shows that my problem is creating the 3x3 areas. Images 4, 5, 6 I held hope that they were correct as the lab description detailed the halftone images would be black and white blocks to recreate a grayscale image. But the halftone image showed it should have been a bit more complex than that. Going forward, I will try to revise my code to obtain the correct pattern for the images.

Results

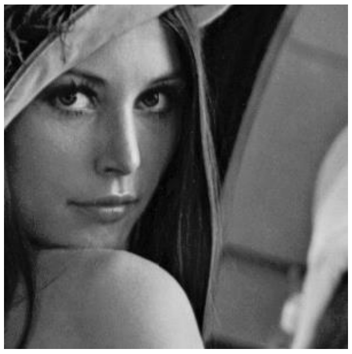
Figure 1: Normal Face Image

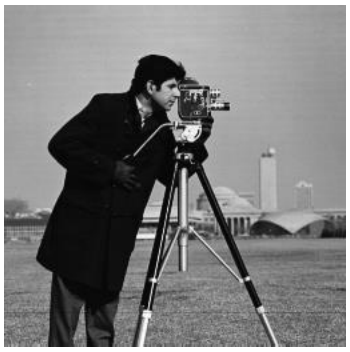
Figure 2: Normal Cameraman Image

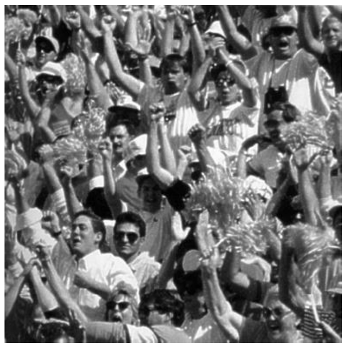
Figure 3: Normal Crowd Image

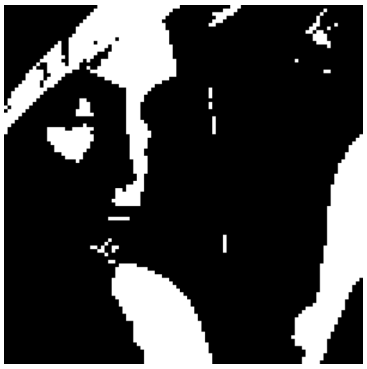
Figure 4: Halftone Face Image

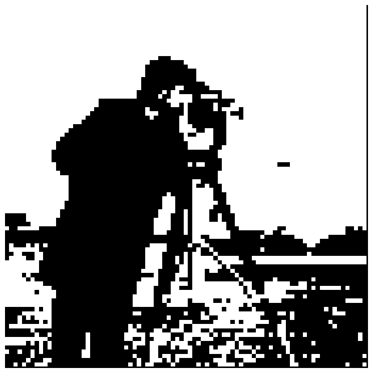
Figure 5: Halftone Cameraman Image

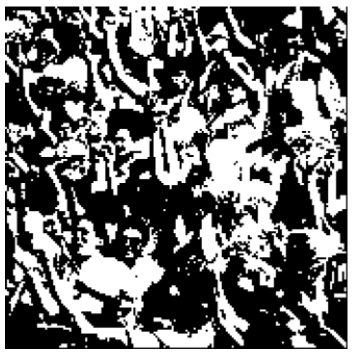
Figure 6: Halftoned Crowd Image

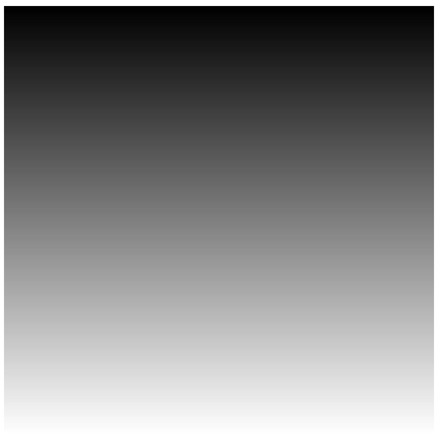
Figure 5: Normal Wedge

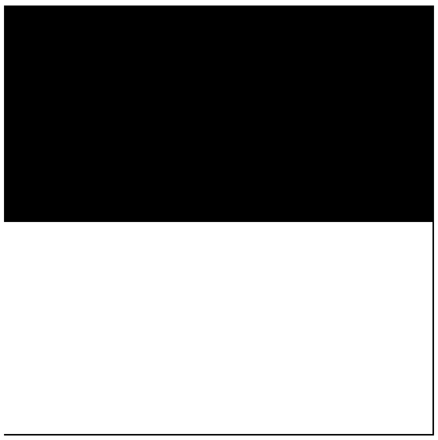
Figure 6: Halftone Wedge

Code

halftone.m

```
%halftone:   Converts a grayscale image to a binary image using
%            black and white dot patterns to render a grayscale
%            image.
%
%Syntax:
%   output_img = halftone(image)
%
%Input:
%   image = The grayscale image that will be rendered. Should
%   be of type uint8 with values 0 to 255.
%
%Output:
%   output_img = Binary output image that is of type uint8 and
%   contains values only 0 or 255.
%
%History:
%   D. Correa       2/15/2022       Created
%   D. Correa       2/21/2022       Added image size boundary
%   D. Correa       2/22/2022       Added average calculation
%                                   and binary value selection.


function output_img = halftone(image)

%get image size%
[row,col] = size(image);

%allocate output image and ensure it is the same size as input
image%
output_img = uint8(ones(row,col));

%take image size as float for average computation
img_float = double(image);

%will determine left over columns that will help make our output
images
%sizes multiples of 3
row_edge = mod(row,3);
col_edge = mod(col,3);

%take the average of each 3x3 block and determine what binary
value will be
%associated with each block%
for i = 1:3:row-row_edge
```

```
    for j = 1:3:col-col_edge
        avg = mean(mean(img_float(i:i+2,j:j+2)));

        %assign binary value to 3x3 block%
        if avg < 255/2
            output_img(i:i+2,j:j+2) = 0;
        else
            output_img(i:i+2,j:j+2) = 255;
        end
    end
end
end
```

wedgeScript.m

```
%create a matrix of zeros size 256x256%
wedge = zeros(256,256);

%fill each row starting with row 1 with a value 0 to 255
for i = 1:256
    for j = 1:256
        wedge(i,j) = i-1;
    end
end

%show create wedge matrix%
imshow(wedge,[]);
imwrite(uint8(wedge), "Wedge.tif");

%set wedge matrix as image matrix to be used in halftone
function%
image = wedge;

%call function%
output_img = halftone(image);


%show and save output image%
imshow(output_img);
imwrite(output_img, "Wedge_Halftone.tif");
```

imagesScript.m

```
%take in images%
face = imread('Fig0225(a)(face).tif');
cameraman = imread('Fig0225(b)(cameraman).tif');
crowd = imread('Fig0225(c)(crowd).tif');

%image to be used for function%
image = face;

%call function%
output_img = halftone(image);

%show output image%
imshow(output_img);

%save images%
imwrite(output_img, "Face_Halftone.tif");
%imwrite(output_img, "Cameraman_Facetone.tif");
%imwrite(output_img, "Crowd_Facetone.tif");
```