Binary Image Rendering using Halftoning

Jose Luciano

February 24, 2022

Abstract

My goal in this lab is to generate a binary image (black and white) from a given greyscale image. To accomplish this I calculated the average intensity value and used it to determine a block pattern. the algorithm was able to successfully convert most of the greyscale images into binary images. However, there are a few exceptions where the algorithm fails such as when the image has dimensions that are not a multiple of 3.

Technical Discussion

The function halftone is able to convert a greyscale image into a binary image. The function traverses the image with 3x3 pixel blocks, and it also calculates the average intensity values of those 3x3 pixel blocks. The average intensity value is then used to determine a specific dot pattern which is used to convert the greyscale image into a binary image.

Similarly, the algorithm attempts to traverse pixel blocks that aren't 3x3 but has difficulties determining the correct dot pattern for conversion. For example, if there exists a 2x3 pixel block the algorithm will attempt to take the average intensity value of that area and determine a dot pattern that best fits that 2x3 pixel block. Lastly, the output image is of the same size as our input.

Discussion of Results

The first image that was used to test the halftone function was Figure 1a. The result of our conversion was to be expected because in the original image we see varying intensity values but in Figure 1b, there are only two, black and white.

Now let us take a look at Figure 2a and its result Figure 2b. Surprisingly, our function is able to convert the greyscale image into a binary image. The quality of our binary image is of lower quality but that is to be expected since we are only working with two intensity values. In this case, our function performed well but that cannot be said for Figures 3a and 4a.

Figures 3a and 4a are of dimensions 256x256 and 512x512 respectively. What this means is that we now have to consider cases where we will have 2x3, 1x3, 3x2, and 3x1 blocks. Unfortunately, by taking a closer look at Figures 3b and 4b we will notice that at the very edge of our binary images there are horizontal black lines. Our algorithm successfully handles the column cases but not the row ones.

Finally, in images with a lesser amount of details, it can be seen that there is a bit of false contouring which is caused by the use of insufficient gray levels. This effect is most noticeable in Figure 2b.

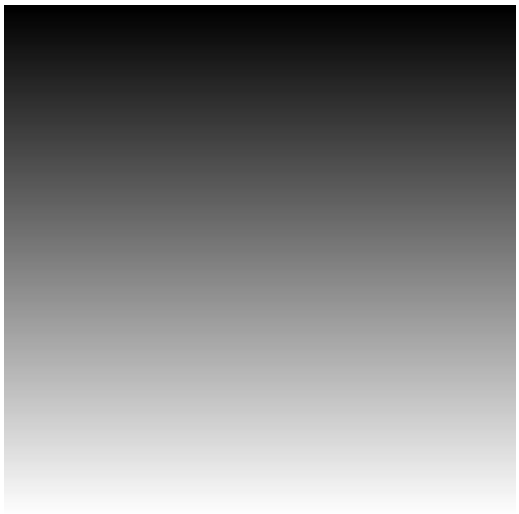

Figure 1a. Wedge image

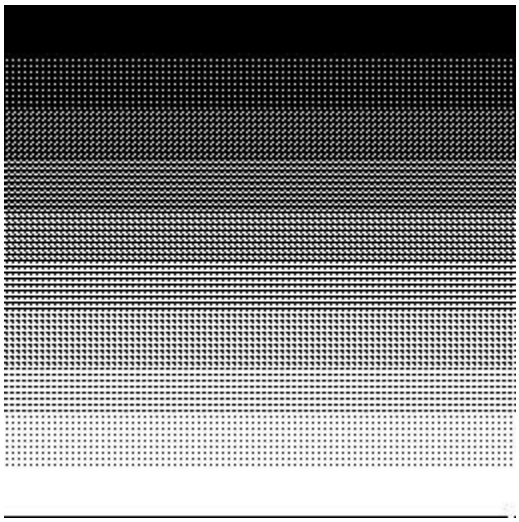

Figure 1b. Halftone conversion of the wedge image

Figure 2a. Face image

Figure 2b. Halftone conversion of the face image

Figure 3a. Cameraman image

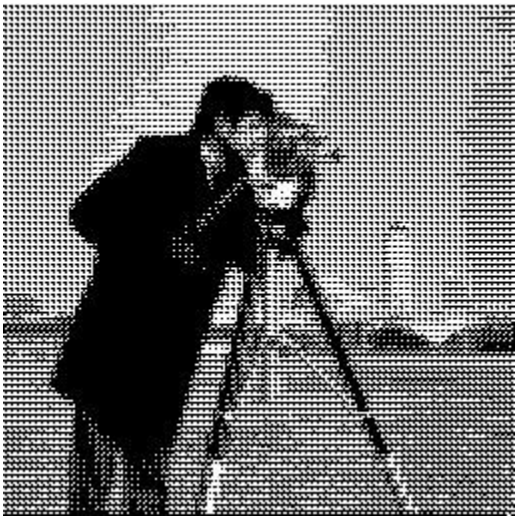

Figure 3b. Halftone conversion of cameraman image

Figure 4a. Crowd image

Figure 4b. Halftone conversion of crowd image

halftone.m

```
function out = halftone(in)
% halftone  The function converts a greyscale image into a binary
 image
%           using dot patterns.
%
% Syntax:
%   out = halftone(in)
%
% Input:
%   in = the grayscale image to be converfted, and should be of type
 uint8
%   with values in the range 0-255.
%
% Output:
%   out = the binary image that will be outputted. It is of type uint8
 and will consist of two values: 0 and
%         255.
% History:
%   J.Luciano - Created function halftone 2/21/2022
%   J.Luciano - Attempted to add edge cases 2/23/2022
%

%create out with size of in
[r c] = size(in);
out = uint8(ones(r, c));
%init dot patterns, 1 to 10
dots = uint8(zeros(3, 3, 10));

dots(:, :, 1) = [0 0 0; 0 0 0; 0 0 0];
dots(:, :, 2) = [0 255 0; 0 0 0; 0 0 0];
dots(:, :, 3) = [0 255 0; 0 0 0; 0 0 255];
dots(:, :, 4) = [255 255 0; 0 0 0; 0 0 255];
dots(:, :, 5) = [255 255 0; 0 0 0; 255 0 255];
dots(:, :, 6) = [255 255 255; 0 0 0; 255 0 255];
dots(:, :, 7) = [255 255 255; 0 0 255; 255 0 255];
dots(:, :, 8) = [255 255 255; 0 0 255; 255 255 255];
dots(:, :, 9) = [255 255 255; 255 0 255; 255 255 255];
dots(:, :, 10) = [255 255 255; 255 255 255; 255 255 255];

in_float = double(in);

%check edges with mod
c_edge = mod(r, 3);
r_edge = mod(c, 3);

%loop whole input image 3x3 pixels -> whats the dot pattern we want in
%the out-> change the 3x3 pixels of output with the dot pattern
for  r1 = 1:3:r-r_edge
    for c1 = 1:3:c-c_edge
        average = mean(mean(in_float(r1:r1+2,c1:c1+2)));
```

```
            %find the dot pattern(:, :, i)
            i = floor((average/255)*10);
            %set i = 9 because we do i+1 to avoid cases where i = 0
                if i == 10
                    i = 9;
                end
            out(r1:r1+2, c1:c1+2) = dots(:,:,i+1);
        end
        if c_edge > 0
            %use the remainder c to calculate the average of a 3x1 or 3x2
     block
            average = mean(mean(in_float(r1:r1+2,c-c_edge+1:c)));
            i = floor((average/255)*10);
            if i == 10
                i = 9;
            end
            out(r1:r1+2, c-c_edge+1:c) = dots(:,1:c_edge,i+1);
        end
        if r_edge > 0
            %use the remainder r to calculate the average of a 1x3 or 2x3
     block
            average = mean(mean(in_float(r-r_edge+1:r,c1:c1+2)));
            %function floor() rounds the value to the nearest integer
            i = floor((average/255)*10);
            if i == 10
                i = 9;
            end
            out(r-r_edge+1:r, c1:c1+2) = dots(1:r_edge,:,i+1);
        end
    end
```

Published with MATLAB® R2020b

testwedge.m

```
test_img = uint8(zeros(256,256));

for c = 1:256
    %0 means black
    test_img(c, :) = c-1; % 0 to 255
end
imshow(test_img);

out = halftone(test_img);

figure;

imshow(out);

imwrite(out, 'test.tif')
imwrite(test_img, 'wedge.tif')
```

Published with MATLAB® R2020b

testpattern.m

```
test_img = imread('Fig0225(a)(face).tif');
imshow(test_img);
out = halftone(test_img);
figure;
imshow(out);
imwrite(out, 'face.tif')


figure;
test_img2 = imread('Fig0225(b)(cameraman).tif');
imshow(test_img2);
out2 = halftone(test_img2);
figure;
imshow(out2);
imwrite(out2, 'cameraman.tif')

figure;
test_img3 = imread('Fig0225(c)(crowd).tif');
imshow(test_img3);
out3 = halftone(test_img3);
figure;
imshow(out3);
imwrite(out3, 'crowd.tif')
```

Published with MATLAB® R2020b