Binary Image Rendering using Halftoning

Sigi Lopez

09/17/18

Halftoning is a technique that simulates a continuous image while using less data than the original Image. The process for Halftoning involves simplifying the data of an Image to binary encoding. Patterns are then used in combination with binary encoding to create a more efficient Image. The benefits of Halftoning is using less data while at the same time be able to see the same image. One of the drawbacks is that you start losing data which means some of the detail of the image gets lost.

Technical Discussion

$$\mathbf{A}\boldsymbol{verage} = \frac{\boldsymbol{Total\ adition\ of\ pixel\ values\ in\ a\ 3x3}}{\boldsymbol{Number\ of\ values}}$$

The approach for making a successful Halftoning function is very simple. The first objective was to differentiate between the range of a fully dynamic grayscale compared to halftoning. To convert grayscale into halftone you first need to take the average of a specific block or area which you want to generalize. After taking the average this tells you how you can connect it to the patterns it relates to the most. For our implementation we used 10 patterns where each of them where a 3x3 pixel block. Each block had only two values, one to represent white and the other to represent black. The advantage of using these 10 patterns is that you can easily implement them to images that are related to the 3x3 aspect ratio. If you try to implement on something that's not 3x3 you might not be able to cover all of the pixels. To solve this problem, I used the ceil function which solves the problem of leaving behind pixels by making them integers instead of decimal numbers on an index. Overall this approach is acceptable since you are already generalizing the image with Halftoning.

Discussion of results

There was a lot of unexpected findings while testing different types of images. The most unexpected results were that some images have different effects depending on how detailed the image is. Images specially with a flatter grayscale where more negatively affected than images that had a wider and more diverse set of values. This can be clearly seen by comparing the images A and D to C and F. C seems to look better than A because the values from the average blocks are still sufficiently diverse to make an illusion as if the Image hasn't lost that much detail. Of Course, you can add more different possibilities to the average which positively adds more detail, but the drawback is that it need more memory to store those values. The parts most affected by halftoning in an image where areas that had a more repetitive tone like a black background or a clear sky since the gradient values in that area where really similar. Overall halftoning is very important specially when it can solve problems like only having two colors such as black and white. Another finding was that applying halftone to an image it made the image more easily noticeable where the averages where picked because there's drastic changes from all the 3x3 blocks. Overall the results of these three images do agree with that's talked in figure 2.26 in the book since you can tell how low-level detail images suffer more degradation with halftoning.

Results

A)

B)

C)

The Images above are halftone versions of the images below. You can easily tell how some images have lost more detail compare to others.

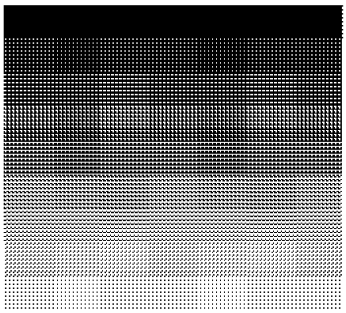
D)

E)

F)

The image below is a halftone of a linear gradient of 256 levels of color. The values were tested from 0 to 255 which is a grayscale from white to black.

G)



Code

```
function [outputImage] = halftone(inputImage)
    %halftone   This function transforms a greyscale image into a binary
image.
    %
    %
    %Input:
    %
    %Its input is an image array/matrix of its grayscale values form 0 to
    %255.
    %
    %Output:
    %
    %Its output is an image array/matrix that only has values 0 and
    %255.
    %
    %This halftone function also uses 10 patterns to simplify the data.
    %To use function simply instert an image Matrix to the function.
    %
    % halftone(InputImageMatrix)   =   Halftoned Image


    %Indexes for input and output image
    [r,c] = size(inputImage);

    %Got rid of possible decimal point for output image (simplified)
    rpro = ceil(r/3)*3;
    cpro = ceil(c/3)*3;

    %Create a
    image = 255*ones(rpro,cpro);
    image(1:r,1:c) = inputImage;


    %Pattern Dimension
    d = 3;

    %Create a matrix that stores the patterns
    pattern = zeros(d,d,d*d+1);

            %Assign matrix values for patterns
            pattern(:,:,10) = [255,255,255; 255,255,255; 255,255,255];

            pattern(:,:,9) = [255,0,255; 255,255,255; 255,255,255];

            pattern(:,:,8) = [255,0,255; 255,255,255; 255,255,0];

            pattern(:,:,7) = [0,0,255; 255,255,255; 255,255,0];

            pattern(:,:,6) = [0,0,255; 255,255,255; 0,255,0];

            pattern(:,:,5) = [0,0,0; 255,255,255; 0,255,0];
```

```
            pattern(:,:,4) = [0,0,0; 255,255,0; 0,255,0];

            pattern(:,:,3) = [0,0,0; 255,255,0; 0,0,0];

            pattern(:,:,2) = [0,0,0; 0,255,0; 0,0,0];

            pattern(:,:,1) = [0,0,0; 0,0,0; 0,0,0];


        %Create

        proImage = zeros(r,c);


        %Loop through whole image
        for(i = 1:rpro/3)
            for(j = 1:cpro/3)

                %Calculate average of selected 3x3 matrix
                average = mean(mean(image((i*3-2):(i*3),(j*3-2):(j*3))));

                %Get Pattern from average
                SelectP = ceil(average/(256/(d*d+1)));

                %Apply pattern to Image
                proImage((i*3-2):(i*3),(j*3-2):(j*3)) = pattern(:,:,
SelectP);
            end
        end

        %Convert image to uint8(values 0-255)
        proImage = uint8(proImage);

        %Copy image for final output
        outputImage = proImage(1:r,1:c);

end




%256 bit wedge

wedge = ones(256,256);

count = 0;

for i = 1:256
    for j = 1:256
        wedge(i,j) = wedge(i,j) + count;
    end
    count = count + 1;
end


image = halftone(wedge);
```

```
imagesc(image)                  % Plot matrix as image
colormap gray;                 % Choose gray level colormap
axis image;                    % Show pixel coordinates as axes
axis off;


%Call function to transform images to halftone
halftoneFace = halftone(imread('Fig0225(a)(face).tif'));
halftoneCameraMan = halftone(imread('Fig0225(b)(cameraman).tif'));
halftoneCrowd = halftone(imread('Fig0225(c)(crowd).tif'));


%Dislpay images that have been halftoned
figure;
subplot(2,2,1),imshow(halftoneFace);
subplot(2,2,2),imshow(halftoneCameraMan);
subplot(2,2,3),imshow(halftoneCrowd);
```