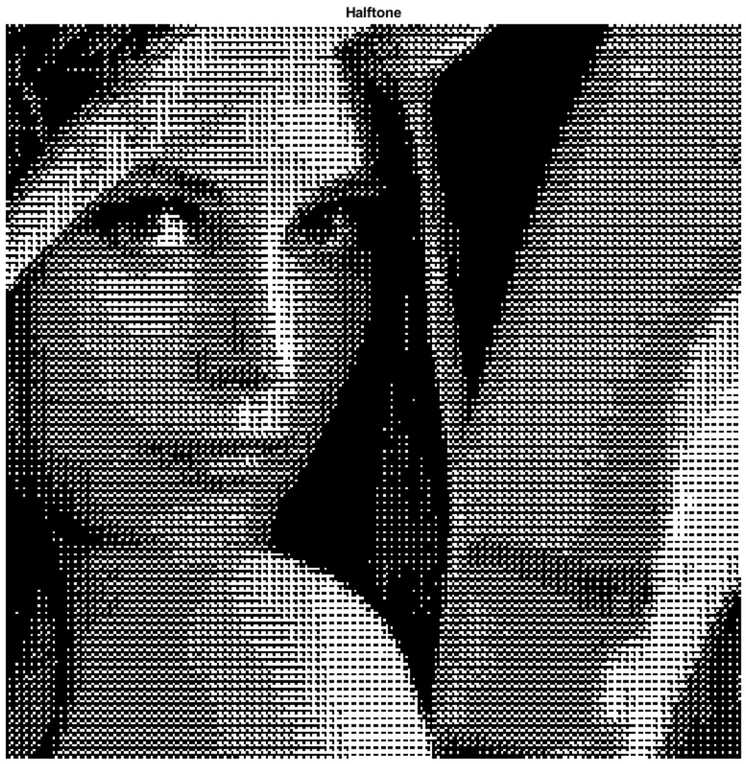


BINARY IMAGE RENDERING USING HALFTONING

February 21, 2022

ABSTRACT

The goal of this lab is to transform a grayscale image into a halftone image. This MATLAB script includes a halftone function that creates matrices containing black and white pixels and uses them to replace pixels in a grayscale image whose values vary. This method outputs an image that appears very similar to the input grayscale image, but only uses black and white pixels and no values in between. This report will discuss the techniques used to successfully create halftone transformations and what each output image tells us about the performance of the program.

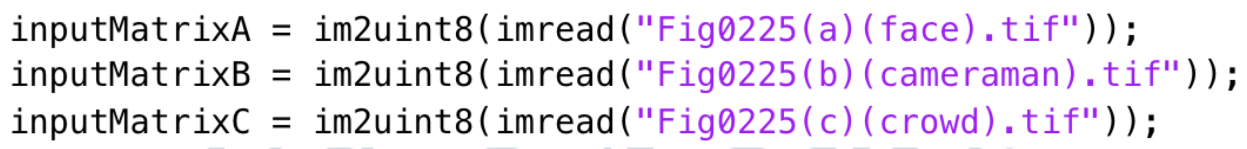
AUTHOR: ALEJANDRO SANCHEZ

CSE 107 – Digital Image Processing

Technical Discussion

How Images are Loaded into the Program

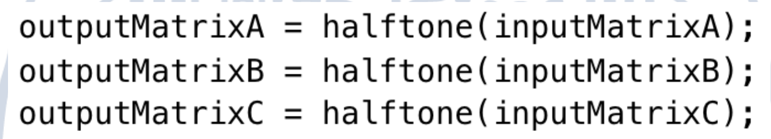
The program starts by reading an image using the `imread()` function. The images that are used in this lab include “`Fig0225(a)(face).tif`”, “`Fig0225(b)(cameraman).tif`”, and “`Fig0225(c)(crowd).tif`”. The image files are converted to the `uint8` type and assigned to a matrix, as shown below in Figure 1.



```
inputMatrixA = im2uint8(imread("Fig0225(a)(face).tif"));
inputMatrixB = im2uint8(imread("Fig0225(b)(cameraman).tif"));
inputMatrixC = im2uint8(imread("Fig0225(c)(crowd).tif"));
```

Figure 1: Importing images and type casting to type uint8

After the matrices are created, they are sent to the `halftone` function as arguments where they will be processed by an algorithm that performs the halftone transformation. In Figure 2, matrices are set equal to the result of the `halftone` function. The result is a matrix of type `logical`.

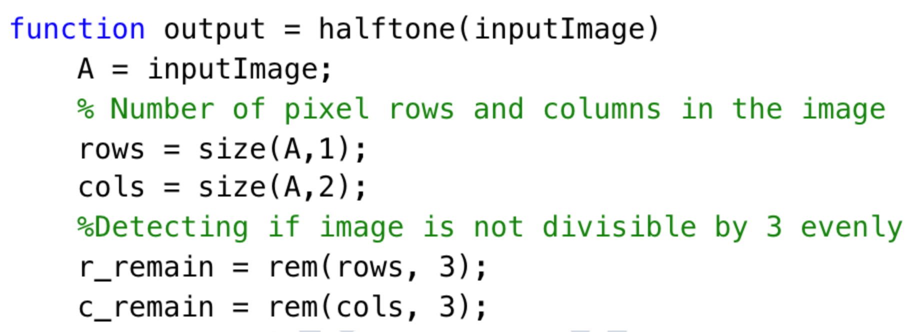


```
outputMatrixA = halftone(inputMatrixA);
outputMatrixB = halftone(inputMatrixB);
outputMatrixC = halftone(inputMatrixC);
```

Figure 2: Assigning matrices as the halftone function’s resulting transformation

The Structure of the Halftone Function

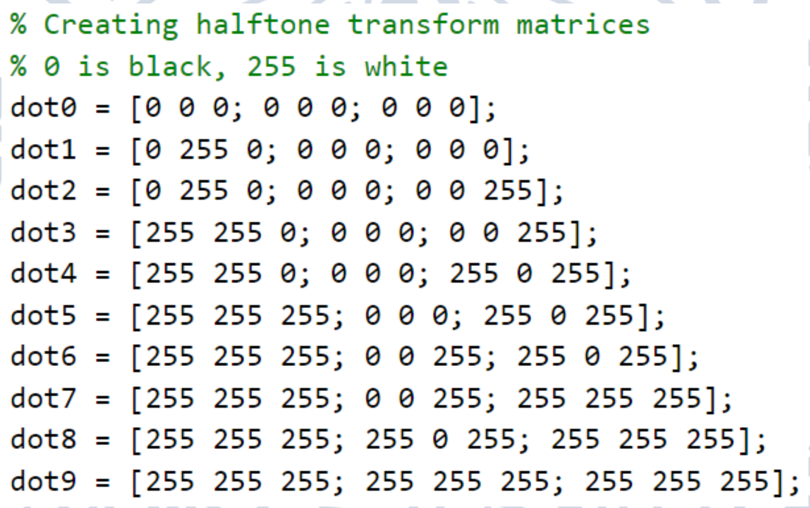
This section will cover the workflow and methodology of the `halftone` function. The function begins by assigning the input image matrix – called `inputImage` in the example shown in the figure below – to a matrix `A`. Next, the function proceeds to identify the dimensions of the image matrix using the `size()` function. More information on how the `size()` function operates on matrices can be found in the appendix. The number of pixels in the image’s x axis is identified as `size(A,1)`, and is assigned to a variable called `rows`. Similarly, the number of pixels in the image’s y axis is found using `size(A,2)`, and is assigned to another variable called `cols`. Since the image may be of any size, the number of pixels that will be left over after performing the halftone transformation must be considered. The remaining values in the x- and y-axes are found using the `rem()` function, and stored into variables `r_remain` and `c_remain`, respectively. These dimensions will be useful later when the function performs the halftoning calculations.



```
function output = halftone(inputImage)
    A = inputImage;
    % Number of pixel rows and columns in the image
    rows = size(A,1);
    cols = size(A,2);
    %Detecting if image is not divisible by 3 evenly
    r_remain = rem(rows, 3);
    c_remain = rem(cols, 3);
```

FIGURE 3: THE START OF THE HALFTONE FUNCTION AND IMAGE DATA COLLECTION

Next, the halftone function proceeds to create 10 arrays called `dot9`, `dot8`, and subsequently `dot0`, in this order, as shown in Figure 4. These arrays are intended to represent raster images, or bitmaps, where a value of 0 is supposed to represent a black pixel, and a 255 representing a white pixel. These arrays represent a halftone cell that will be assigned to portions of a grayscale image.



```
% Creating halftone transform matrices
% 0 is black, 255 is white
dot0 = [0 0 0; 0 0 0; 0 0 0];
dot1 = [0 255 0; 0 0 0; 0 0 0];
dot2 = [0 255 0; 0 0 0; 0 0 255];
dot3 = [255 255 0; 0 0 0; 0 0 255];
dot4 = [255 255 0; 0 0 0; 255 0 255];
dot5 = [255 255 255; 0 0 0; 255 0 255];
dot6 = [255 255 255; 0 0 255; 255 0 255];
dot7 = [255 255 255; 0 0 255; 255 255 255];
dot8 = [255 255 255; 255 0 255; 255 255 255];
dot9 = [255 255 255; 255 255 255; 255 255 255];
```

FIGURE 4: ARRAYS ARE CREATED TO REPRESENT HALFTONE OUTPUT CELLS

The goal of the grayscale image being assigned these bitmap array values is that it will only have pixel values of 0 or 255. When it is then converted to a binary image of logical type, the result will have 1s and 0s where 0s represent black pixels and 1s represent white pixels. These bitmap arrays have different quantities of black pixels (0-pixel value) because they are supposed to represent raster images, as shown below in Figure 5. The ratio of the black areas to the non-black areas of the raster image corresponds to the luminance of an input cell from the grayscale image. The goal of the halftone function is to produce a binary image (1s and 0s) that appears like the original grayscale image from afar. The black pixels in the bitmap arrays are static and will not move. Additionally, the quantity of black pixels will not change.

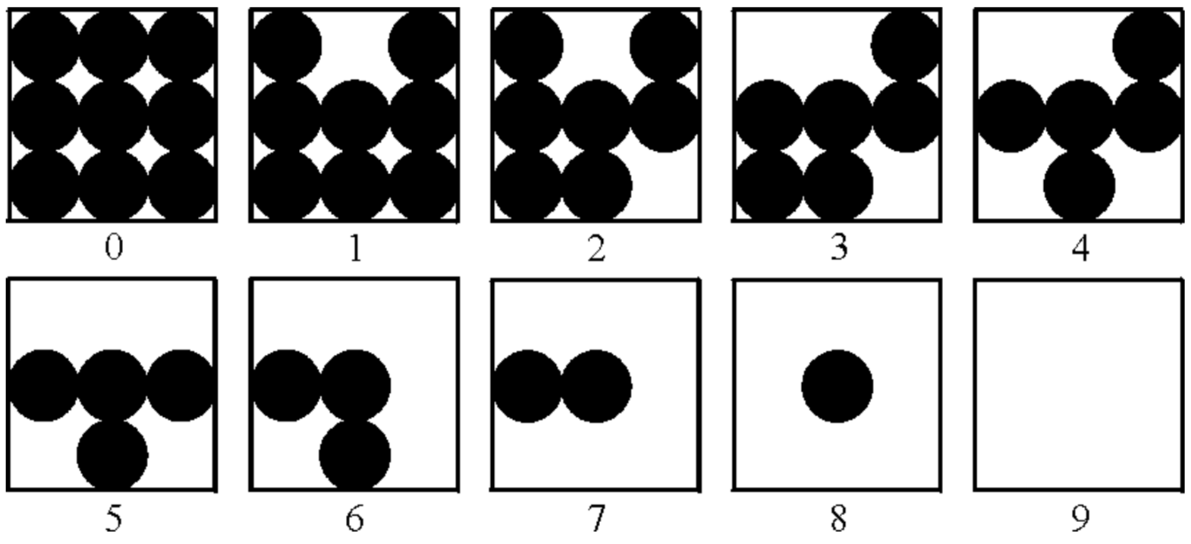
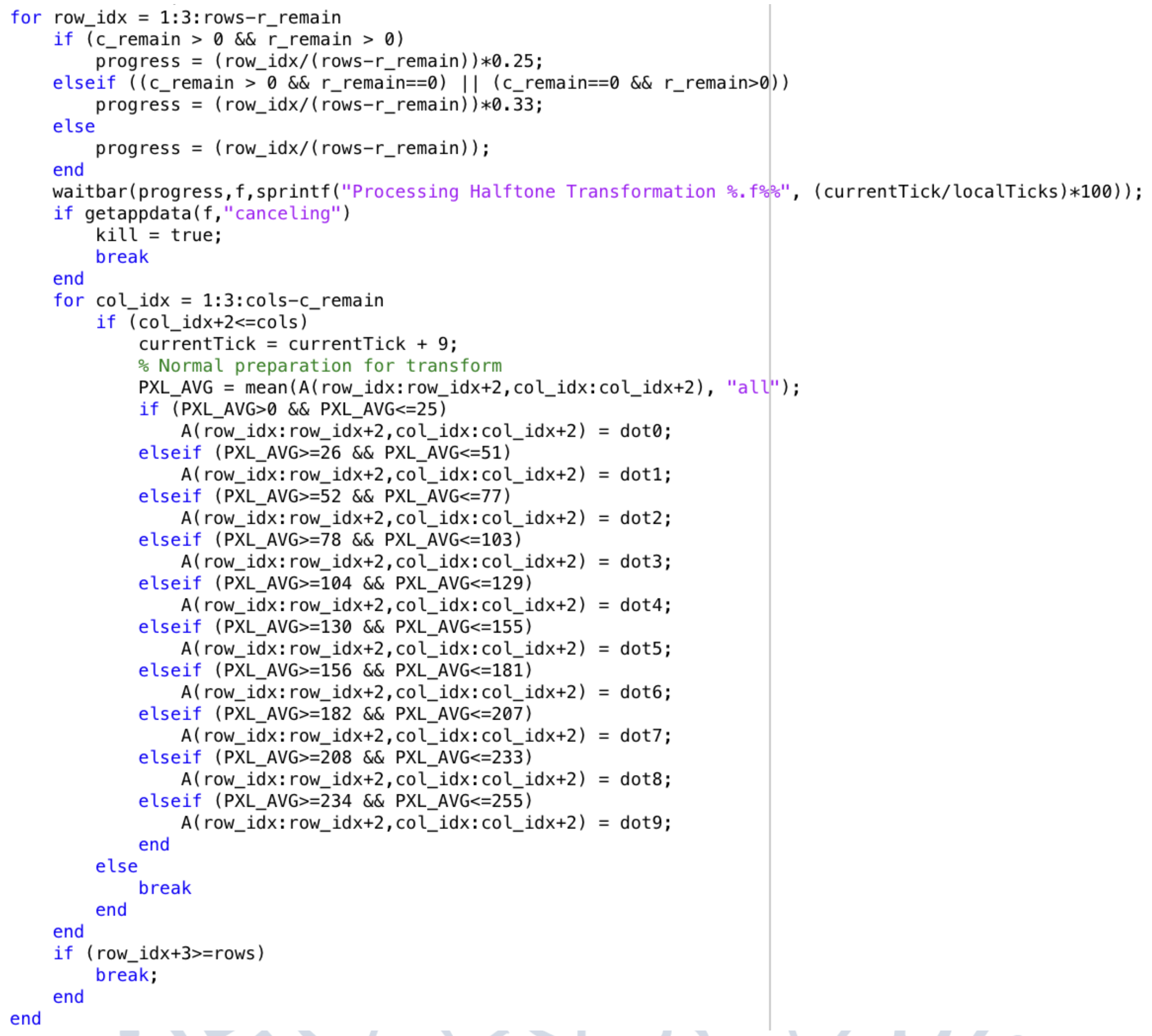


FIGURE 5: RASTER IMAGES OF BLACK (0 GRAYSCALE VALUE) AND WHITE (255 GRAYSCALE VALUE) PIXELS

After the bitmap arrays are created, the function proceeds to perform the halftone transformation in four different patterns. The first pattern is going from the top of the image, moving from left to right, and moving down towards the bottom. This pattern iterates through the number of pixels in the x- and y-axis in intervals of three. Since the image is assumed to have any type of dimensions, the pattern will move from left to right until the next interval of three pixels is detected to go out of the image's boundaries (right edge). When this boundary detection is made, or until the intervals have met the end of the image on its right edge, the pattern resets back at the left side if another interval of three pixels may continue towards the bottom of the image. If the next interval of three pixels is detected to exceed the image's boundary in the x-axis (bottom edge), no more transformations will occur, and this pattern ends. The first pattern, labeled "UPPER-LEFT LEFT-TO-RIGHT TRANSFORM" is shown below in Figure 6.

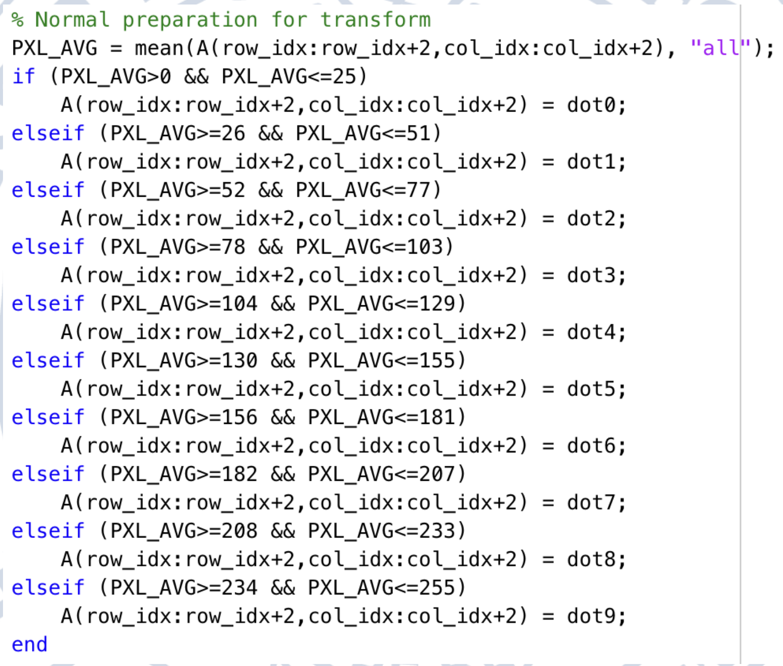


```
for row_idx = 1:3:rows-r_remain
    if (c_remain > 0 && r_remain > 0)
        progress = (row_idx/(rows-r_remain))*0.25;
    elseif ((c_remain > 0 && r_remain==0) || (c_remain==0 && r_remain>0))
        progress = (row_idx/(rows-r_remain))*0.33;
    else
        progress = (row_idx/(rows-r_remain));
    end
    waitbar(progress,f,sprintf("Processing Halftone Transformation %.f%%", (currentTick/localTicks)*100));
    if getappdata(f,"canceling")
        kill = true;
        break
    end
    for col_idx = 1:3:cols-c_remain
        if (col_idx+2<=cols)
            currentTick = currentTick + 9;
            % Normal preparation for transform
            PXL_AVG = mean(A(row_idx:row_idx+2,col_idx:col_idx+2), "all");
            if (PXL_AVG>0 && PXL_AVG<=25)
                A(row_idx:row_idx+2,col_idx:col_idx+2) = dot0;
            elseif (PXL_AVG>=26 && PXL_AVG<=51)
                A(row_idx:row_idx+2,col_idx:col_idx+2) = dot1;
            elseif (PXL_AVG>=52 && PXL_AVG<=77)
                A(row_idx:row_idx+2,col_idx:col_idx+2) = dot2;
            elseif (PXL_AVG>=78 && PXL_AVG<=103)
                A(row_idx:row_idx+2,col_idx:col_idx+2) = dot3;
            elseif (PXL_AVG>=104 && PXL_AVG<=129)
                A(row_idx:row_idx+2,col_idx:col_idx+2) = dot4;
            elseif (PXL_AVG>=130 && PXL_AVG<=155)
                A(row_idx:row_idx+2,col_idx:col_idx+2) = dot5;
            elseif (PXL_AVG>=156 && PXL_AVG<=181)
                A(row_idx:row_idx+2,col_idx:col_idx+2) = dot6;
            elseif (PXL_AVG>=182 && PXL_AVG<=207)
                A(row_idx:row_idx+2,col_idx:col_idx+2) = dot7;
            elseif (PXL_AVG>=208 && PXL_AVG<=233)
                A(row_idx:row_idx+2,col_idx:col_idx+2) = dot8;
            elseif (PXL_AVG>=234 && PXL_AVG<=255)
                A(row_idx:row_idx+2,col_idx:col_idx+2) = dot9;
            end
        else
            break
        end
    end
    if (row_idx+3>=rows)
        break;
    end
end
```

FIGURE 6: FIRST HALFTONE TRANSFORM ALGORITHM

The first transform pattern starts with a `for` loop that goes by each row in intervals of three until the end of the image on its right side. The nested for loop afterwards carries on by going by each column in intervals of three until the end of the image on its bottom side. The intervals of three going in both right and down is what gives this pattern its name "UPPER-LEFT LEFT-TO-RIGHT". The intervals in both directions form arrays of size 3x3. These arrays are selected in the image using element-wise selection, a method in MATLAB that allows the user to select regions inside of an array. This is apparent in the assignment of variable `PXL_AVG`, where `row_idx` is the current row index in the first `for` loop, and by using the character '`:`', the element `row_idx+2` is the last row index of the current interval. Using this technique, the algorithm will select row ranges in the image that are three pixels in size. The same technique is applied when selecting the column ranges, using `col_idx` as the current index, and `col_idx+2` as the last column in the current interval. `PXL_AVG` takes the sum of all pixel values in these 3x3 pixel ranges (rows and columns) and computes the average with the `mean()` function.

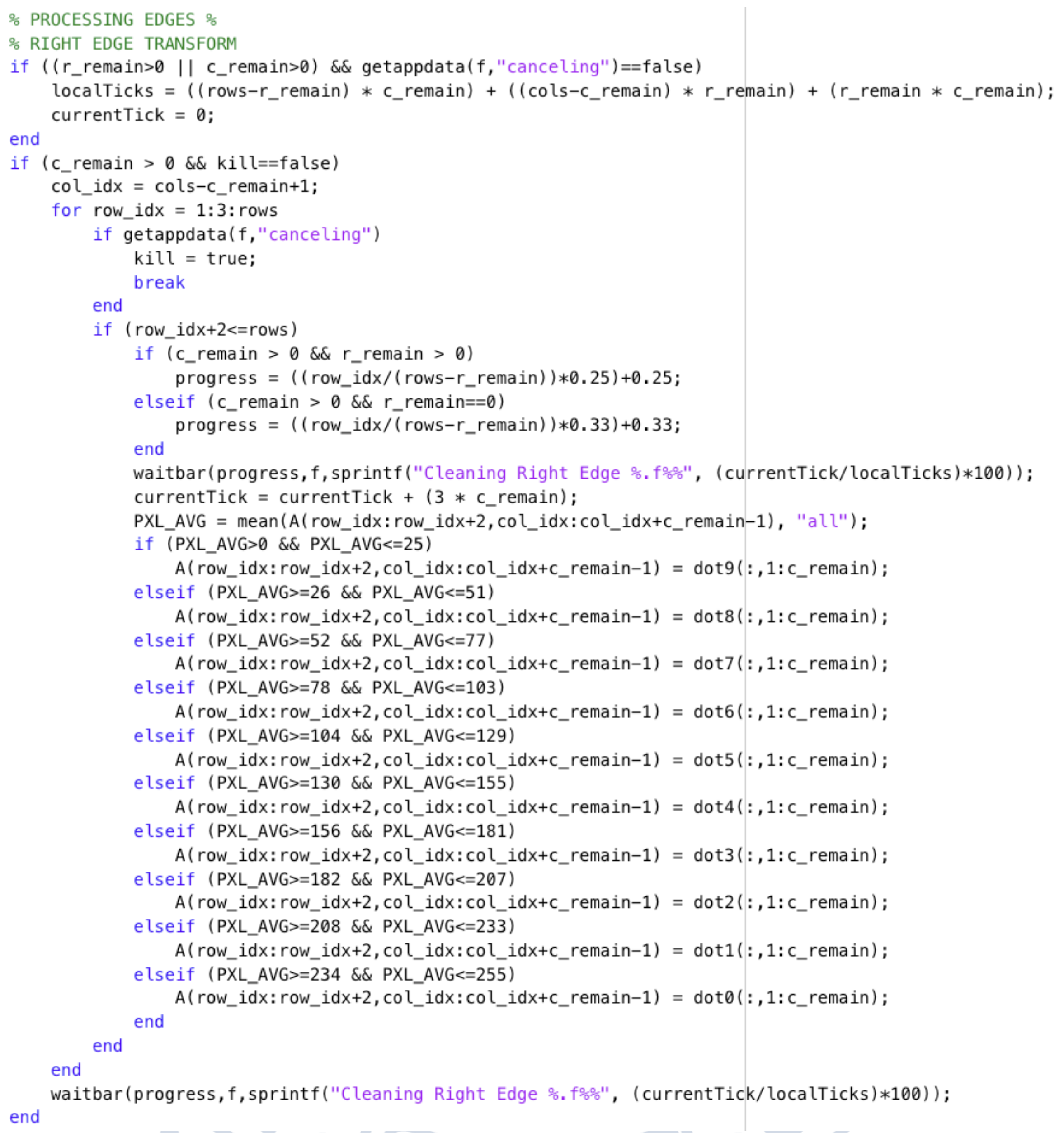
After the average pixel value has been calculated, it is compared to different possible ranges. These ranges, shown below in Figure 7, have specific values. These values were determined by taking the range of pixels that could be assigned to the variable `PXL_AVG` and split up into ten different ranges. There are ten different ranges because there are ten different bitmap arrays. When the value `PXL_AVG` falls into one of these ranges, the algorithm will assign the same row and column range in the two `for` loops to the values of the bitmap. For example, `dot9` is used in the first range because it has no black pixels. The image will be assigned to the values of this bitmap array if the average pixel value is between 0 and 25, which to the average person appears mostly white on a grayscale image. Finally, the image in this row and column range will contain only zeros. After this assignment, the transformation is over and the next column or row in the image is iterated to repeat this calculation and assignment process. Since there are different pixel averages throughout the image, the best-case scenario is that all the conditional statements will be used and all bitmap arrays `dot9` through `dot0` will be used.



```
% Normal preparation for transform
PXL_AVG = mean(A(row_idx:row_idx+2,col_idx:col_idx+2), "all");
if (PXL_AVG>0 && PXL_AVG<=25)
    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot0;
elseif (PXL_AVG>=26 && PXL_AVG<=51)
    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot1;
elseif (PXL_AVG>=52 && PXL_AVG<=77)
    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot2;
elseif (PXL_AVG>=78 && PXL_AVG<=103)
    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot3;
elseif (PXL_AVG>=104 && PXL_AVG<=129)
    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot4;
elseif (PXL_AVG>=130 && PXL_AVG<=155)
    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot5;
elseif (PXL_AVG>=156 && PXL_AVG<=181)
    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot6;
elseif (PXL_AVG>=182 && PXL_AVG<=207)
    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot7;
elseif (PXL_AVG>=208 && PXL_AVG<=233)
    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot8;
elseif (PXL_AVG>=234 && PXL_AVG<=255)
    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot9;
end
```

FIGURE 7: ASSIGNING GRAYSCALE IMAGES TO BITMAP ARRAY VALUES IN FIRST HALFTONE TRANSFORM

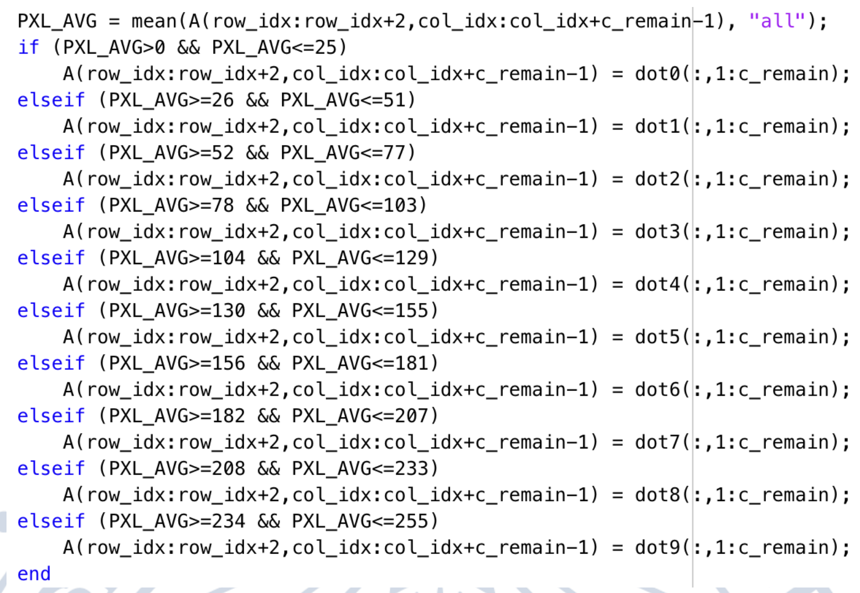
The next pattern in the halftone function is called the “RIGHT EDGE TRANSFORM”, which performs halftone transformations along the right edge of an image if the image’s dimension along the y axis is not divisible evenly by three. In other words, if the value of `c_remain` is greater than zero, it means that there is a remainder of columns after dividing the number of column pixels by three. This suggests that the “UPPER-LEFT LEFT-TO-RIGHT TRANSFORM” pattern would not perform halftone transformations along the right edge.
The algorithm for this second pattern is shown below in Figure 8. It uses the same technique of going through rows and columns without stepping over bounds.



```
% PROCESSING EDGES %
% RIGHT EDGE TRANSFORM
if ((r_remain>0 || c_remain>0) && getappdata(f,"canceling")==false)
    localTicks = ((rows-r_remain) * c_remain) + ((cols-c_remain) * r_remain) + (r_remain * c_remain);
    currentTick = 0;
end
if (c_remain > 0 && kill==false)
    col_idx = cols-c_remain+1;
    for row_idx = 1:3:rows
        if getappdata(f,"canceling")
            kill = true;
            break
        end
        if (row_idx+2<=rows)
            if (c_remain > 0 && r_remain > 0)
                progress = ((row_idx/(rows-r_remain))*0.25)+0.25;
            elseif (c_remain > 0 && r_remain==0)
                progress = ((row_idx/(rows-r_remain))*0.33)+0.33;
            end
            waitbar(progress,f,sprintf("Cleaning Right Edge %.f%%", (currentTick/localTicks)*100));
            currentTick = currentTick + (3 * c_remain);
            PXL_AVG = mean(A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1), "all");
            if (PXL_AVG>0 && PXL_AVG<=25)
                A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot9(:,1:c_remain);
            elseif (PXL_AVG>=26 && PXL_AVG<=51)
                A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot8(:,1:c_remain);
            elseif (PXL_AVG>=52 && PXL_AVG<=77)
                A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot7(:,1:c_remain);
            elseif (PXL_AVG>=78 && PXL_AVG<=103)
                A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot6(:,1:c_remain);
            elseif (PXL_AVG>=104 && PXL_AVG<=129)
                A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot5(:,1:c_remain);
            elseif (PXL_AVG>=130 && PXL_AVG<=155)
                A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot4(:,1:c_remain);
            elseif (PXL_AVG>=156 && PXL_AVG<=181)
                A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot3(:,1:c_remain);
            elseif (PXL_AVG>=182 && PXL_AVG<=207)
                A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot2(:,1:c_remain);
            elseif (PXL_AVG>=208 && PXL_AVG<=233)
                A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot1(:,1:c_remain);
            elseif (PXL_AVG>=234 && PXL_AVG<=255)
                A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot0(:,1:c_remain);
            end
        end
    end
    waitbar(progress,f,sprintf("Cleaning Right Edge %.f%%", (currentTick/localTicks)*100));
end
```

FIGURE 8: SECOND HALFTONE TRANSFORM ALGORITHM

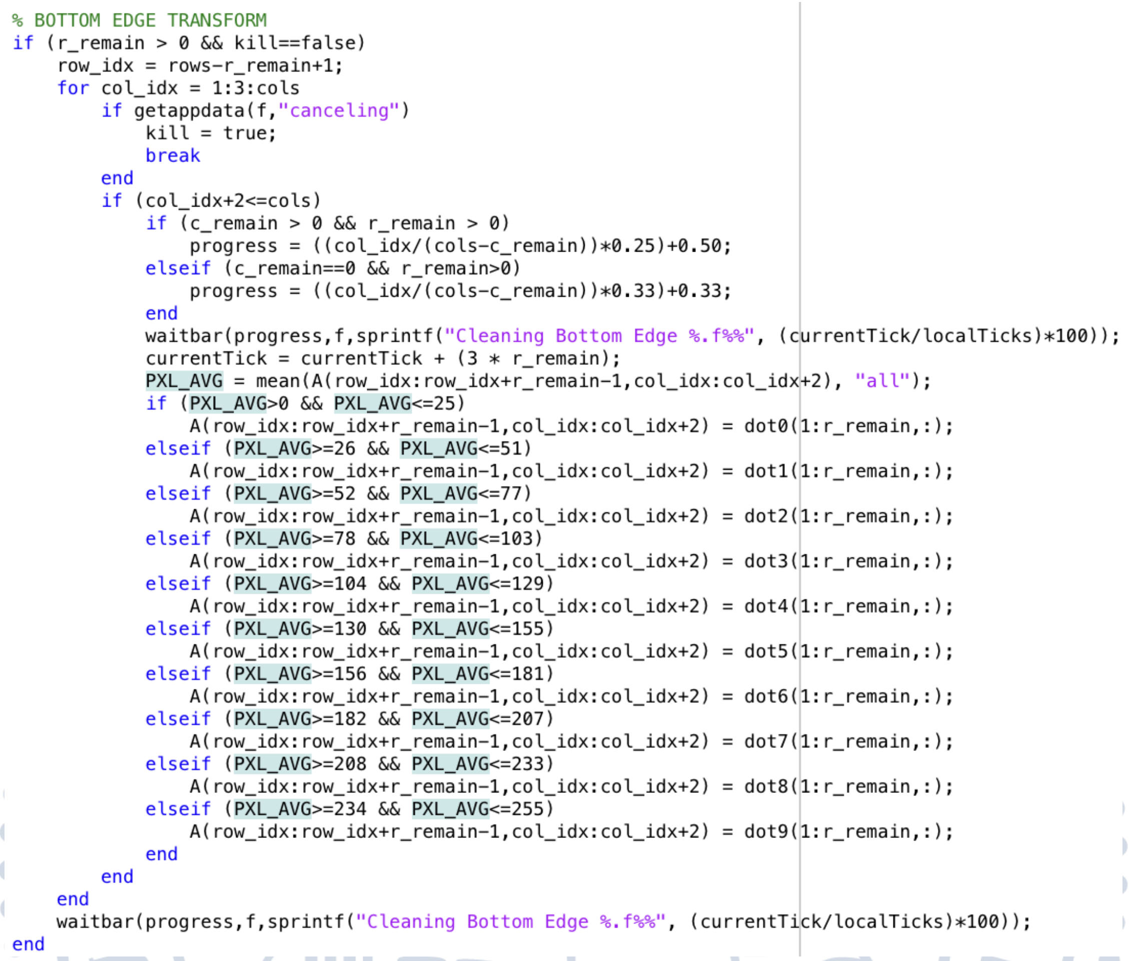
At the start of the second halftone transform algorithm, the value of `col_idx` is equal to the start of the remaining range that needs to be iterated through. In other words, since the first algorithm covered a set of indices of multiples of three, this second algorithm will cover the remainder of columns on the y-axis. To do so, the value of `col_idx` is set equal to the maximum number of columns (`cols`) minus the remainder of columns after performing the remainder function (`rem()`) in the beginning of the halftone function (`c_remain`), plus one – to account for indices starting at ‘`1`’ in MATLAB, not ‘`0`’. In Figure 9 below, the methodology for assigning bitmap arrays to ranges in the grayscale image is shown for every average range of pixels.



```
PXL_AVG = mean(A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1), "all");
if (PXL_AVG>0 && PXL_AVG<=25)
    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot0(:,1:c_remain);
elseif (PXL_AVG>=26 && PXL_AVG<=51)
    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot1(:,1:c_remain);
elseif (PXL_AVG>=52 && PXL_AVG<=77)
    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot2(:,1:c_remain);
elseif (PXL_AVG>=78 && PXL_AVG<=103)
    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot3(:,1:c_remain);
elseif (PXL_AVG>=104 && PXL_AVG<=129)
    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot4(:,1:c_remain);
elseif (PXL_AVG>=130 && PXL_AVG<=155)
    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot5(:,1:c_remain);
elseif (PXL_AVG>=156 && PXL_AVG<=181)
    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot6(:,1:c_remain);
elseif (PXL_AVG>=182 && PXL_AVG<=207)
    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot7(:,1:c_remain);
elseif (PXL_AVG>=208 && PXL_AVG<=233)
    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot8(:,1:c_remain);
elseif (PXL_AVG>=234 && PXL_AVG<=255)
    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) = dot9(:,1:c_remain);
end
```

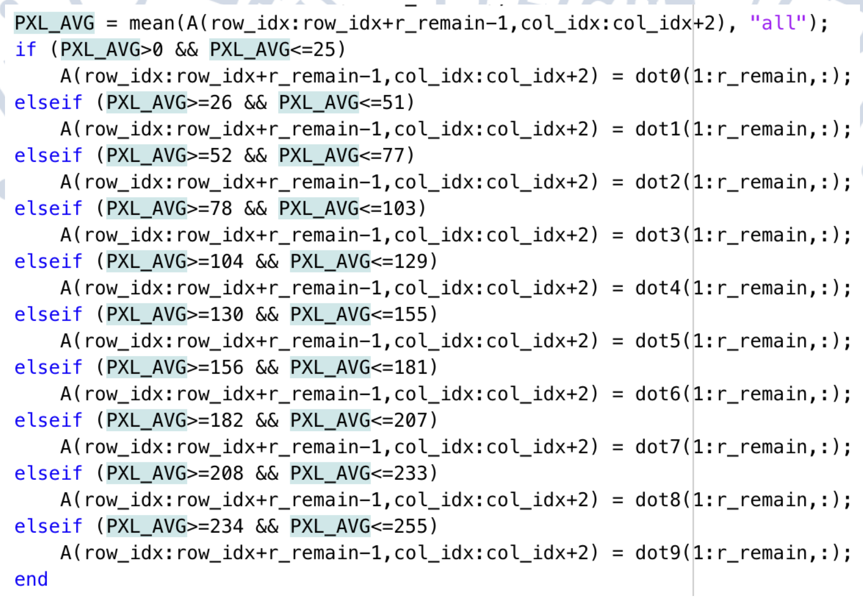
FIGURE 9: ASSIGNING GRAYSCALE IMAGES TO BITMAP ARRAY VALUES IN SECOND HALFTONE TRANSFORM

In the third halftone transform algorithm, named "BOTTOM EDGE TRANSFORM", the bottom edge of the image in the x-axis is considered for any possible remaining pixels that were not covered in the first and second transformation. To accommodate these pixels, the algorithm shown below in Figure 10 utilizes a similar method as the "RIGHT EDGE TRANSFORM" algorithm such that the row and column indices are switched places. The grayscale image is permuted and assigned bitmap array values according to their average pixel values, as shown in Figure 11.



```
% BOTTOM EDGE TRANSFORM
if (r_remain > 0 && kill==false)
    row_idx = rows-r_remain+1;
    for col_idx = 1:3:cols
        if getappdata(f,"canceling")
            kill = true;
            break
        end
        if (col_idx+2<=cols)
            if (c_remain > 0 && r_remain > 0)
                progress = ((col_idx/(cols-c_remain))*0.25)+0.50;
            elseif (c_remain==0 && r_remain>0)
                progress = ((col_idx/(cols-c_remain))*0.33)+0.33;
            end
            waitbar(progress,f,sprintf("Cleaning Bottom Edge %.f%%", (currentTick/localTicks)*100));
            currentTick = currentTick + (3 * r_remain);
            PXL_AVG = mean(A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2), "all");
            if (PXL_AVG>0 && PXL_AVG<=25)
                A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot0(1:r_remain,:);
            elseif (PXL_AVG>=26 && PXL_AVG<=51)
                A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot1(1:r_remain,:);
            elseif (PXL_AVG>=52 && PXL_AVG<=77)
                A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot2(1:r_remain,:);
            elseif (PXL_AVG>=78 && PXL_AVG<=103)
                A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot3(1:r_remain,:);
            elseif (PXL_AVG>=104 && PXL_AVG<=129)
                A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot4(1:r_remain,:);
            elseif (PXL_AVG>=130 && PXL_AVG<=155)
                A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot5(1:r_remain,:);
            elseif (PXL_AVG>=156 && PXL_AVG<=181)
                A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot6(1:r_remain,:);
            elseif (PXL_AVG>=182 && PXL_AVG<=207)
                A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot7(1:r_remain,:);
            elseif (PXL_AVG>=208 && PXL_AVG<=233)
                A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot8(1:r_remain,:);
            elseif (PXL_AVG>=234 && PXL_AVG<=255)
                A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot9(1:r_remain,:);
            end
        end
    end
    waitbar(progress,f,sprintf("Cleaning Bottom Edge %.f%%", (currentTick/localTicks)*100));
end
```

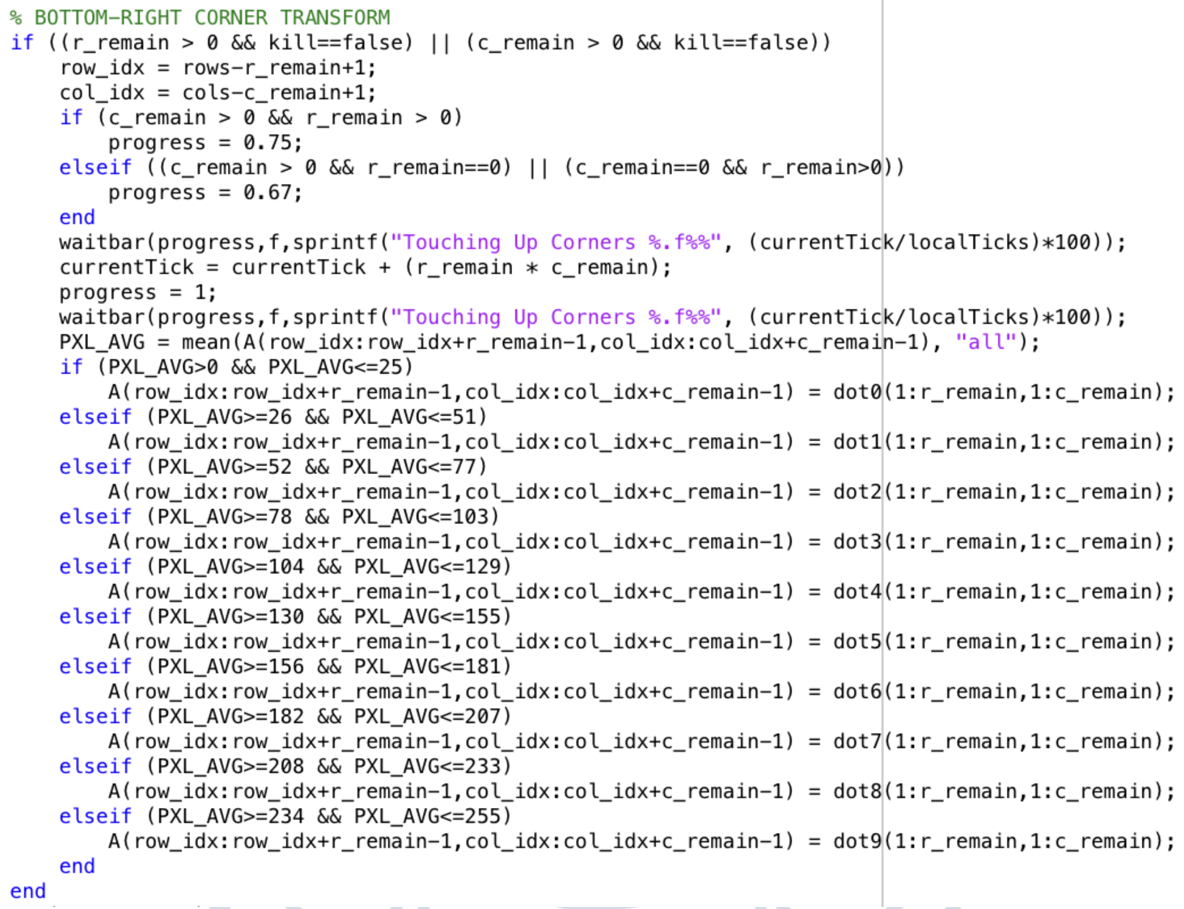
FIGURE 10: THIRD HALFTONE TRANSFORM ALGORITHM



```
PXL_AVG = mean(A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2), "all");
if (PXL_AVG>0 && PXL_AVG<=25)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot0(1:r_remain,:);
elseif (PXL_AVG>=26 && PXL_AVG<=51)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot1(1:r_remain,:);
elseif (PXL_AVG>=52 && PXL_AVG<=77)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot2(1:r_remain,:);
elseif (PXL_AVG>=78 && PXL_AVG<=103)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot3(1:r_remain,:);
elseif (PXL_AVG>=104 && PXL_AVG<=129)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot4(1:r_remain,:);
elseif (PXL_AVG>=130 && PXL_AVG<=155)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot5(1:r_remain,:);
elseif (PXL_AVG>=156 && PXL_AVG<=181)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot6(1:r_remain,:);
elseif (PXL_AVG>=182 && PXL_AVG<=207)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot7(1:r_remain,:);
elseif (PXL_AVG>=208 && PXL_AVG<=233)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot8(1:r_remain,:);
elseif (PXL_AVG>=234 && PXL_AVG<=255)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) = dot9(1:r_remain,:);
end
```

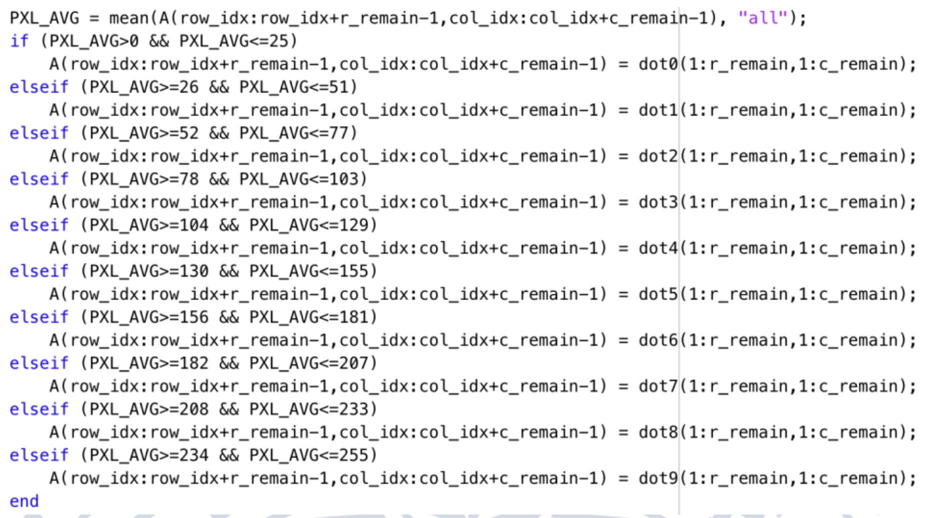
FIGURE 11: ASSIGNING GRAYSCALE IMAGES TO BITMAP ARRAY VALUES IN THIRD HALFTONE TRANSFORM

The fourth and final algorithm that handles the grayscale to halftone process is called the "BOTTOM-RIGHT CORNER TRANSFORM" pattern. The algorithm functions to accommodate the pixels that were missed by all the previous transformation methods by taking the upper limits of remaining pixel ranges in the x- and y-axis. Like the second and third halftone transform algorithm, the row and column indices of the grayscale image are the upper limits in the "BOTTOM-RIGHT CORNER" algorithm, as shown in Figure 12. These ranges are used to identify the last pixels in the very bottom-right corner of the image, forming a very small rectangle or square – assuming there are remainders in the x- or y-axis. The bitmap arrays are set to ranges in the grayscale image respective to the average pixel values, as shown in Figure 13.



```
% BOTTOM-RIGHT CORNER TRANSFORM
if ((r_remain > 0 && kill==false) || (c_remain > 0 && kill==false))
    row_idx = rows-r_remain+1;
    col_idx = cols-c_remain+1;
    if (c_remain > 0 && r_remain > 0)
        progress = 0.75;
    elseif ((c_remain > 0 && r_remain==0) || (c_remain==0 && r_remain>0))
        progress = 0.67;
    end
    waitbar(progress,f,sprintf("Touching Up Corners %.f%%", (currentTick/localTicks)*100));
    currentTick = currentTick + (r_remain * c_remain);
    progress = 1;
    waitbar(progress,f,sprintf("Touching Up Corners %.f%%", (currentTick/localTicks)*100));
    PXL_AVG = mean(A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1), "all");
    if (PXL_AVG>0 && PXL_AVG<=25)
        A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot0(1:r_remain,1:c_remain);
    elseif (PXL_AVG>=26 && PXL_AVG<=51)
        A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot1(1:r_remain,1:c_remain);
    elseif (PXL_AVG>=52 && PXL_AVG<=77)
        A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot2(1:r_remain,1:c_remain);
    elseif (PXL_AVG>=78 && PXL_AVG<=103)
        A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot3(1:r_remain,1:c_remain);
    elseif (PXL_AVG>=104 && PXL_AVG<=129)
        A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot4(1:r_remain,1:c_remain);
    elseif (PXL_AVG>=130 && PXL_AVG<=155)
        A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot5(1:r_remain,1:c_remain);
    elseif (PXL_AVG>=156 && PXL_AVG<=181)
        A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot6(1:r_remain,1:c_remain);
    elseif (PXL_AVG>=182 && PXL_AVG<=207)
        A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot7(1:r_remain,1:c_remain);
    elseif (PXL_AVG>=208 && PXL_AVG<=233)
        A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot8(1:r_remain,1:c_remain);
    elseif (PXL_AVG>=234 && PXL_AVG<=255)
        A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot9(1:r_remain,1:c_remain);
    end
end
```

FIGURE 12: FOURTH HALFTONE TRANSFORM ALGORITHM



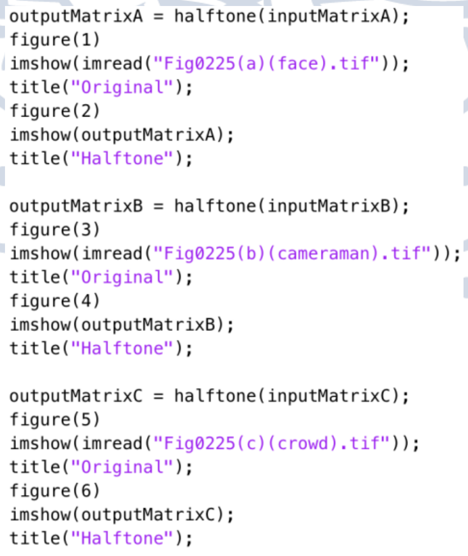
```
PXL_AVG = mean(A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1), "all");
if (PXL_AVG>0 && PXL_AVG<=25)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot0(1:r_remain,1:c_remain);
elseif (PXL_AVG>=26 && PXL_AVG<=51)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot1(1:r_remain,1:c_remain);
elseif (PXL_AVG>=52 && PXL_AVG<=77)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot2(1:r_remain,1:c_remain);
elseif (PXL_AVG>=78 && PXL_AVG<=103)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot3(1:r_remain,1:c_remain);
elseif (PXL_AVG>=104 && PXL_AVG<=129)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot4(1:r_remain,1:c_remain);
elseif (PXL_AVG>=130 && PXL_AVG<=155)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot5(1:r_remain,1:c_remain);
elseif (PXL_AVG>=156 && PXL_AVG<=181)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot6(1:r_remain,1:c_remain);
elseif (PXL_AVG>=182 && PXL_AVG<=207)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot7(1:r_remain,1:c_remain);
elseif (PXL_AVG>=208 && PXL_AVG<=233)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot8(1:r_remain,1:c_remain);
elseif (PXL_AVG>=234 && PXL_AVG<=255)
    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) = dot9(1:r_remain,1:c_remain);
end
```

Figure 13: Assigning grayscale images to bitmap array values in fourth halftone transform

Results and Discussion

Grayscale Images and Halftone Transformations

The three images are originally grayscale images. As discussed before, they are used as arguments for the halftone function in the script. The input type is `uint8` and the output of the halftone function is `uint8` type as well. However, the values in the output image have only one of two values, ‘`0`’ and ‘`255`’, to represent a halftone transformation.
In the main function, the original image is presented to the user and the halftone image is presented after the function takes the original image and processes it. This process is done for all three images and is shown below in Figure 14.



```
outputMatrixA = halftone(inputMatrixA);
figure(1)
imshow(imread("Fig0225(a)(face).tif"));
title("Original");
figure(2)
imshow(outputMatrixA);
title("Halftone");

outputMatrixB = halftone(inputMatrixB);
figure(3)
imshow(imread("Fig0225(b)(cameraman).tif"));
title("Original");
figure(4)
imshow(outputMatrixB);
title("Halftone");

outputMatrixC = halftone(inputMatrixC);
figure(5)
imshow(imread("Fig0225(c)(crowd).tif"));
title("Original");
figure(6)
imshow(outputMatrixC);
title("Halftone");
```

Figure 14: Using original images to call halftone function and displaying processed images

The Halftone Images

The first, second, and third images before their halftone transformation is shown below in Figure 15, 17, and 19, respectively, and their halftone versions are labeled 16, 18, and 20, respectively.

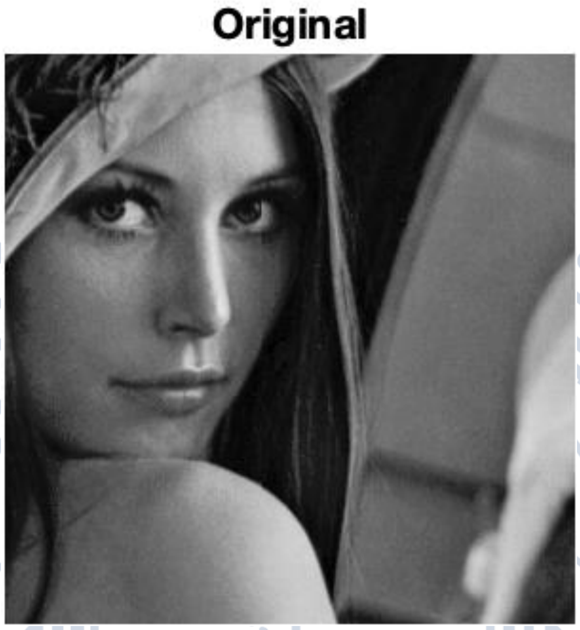


Figure 15: Original face image

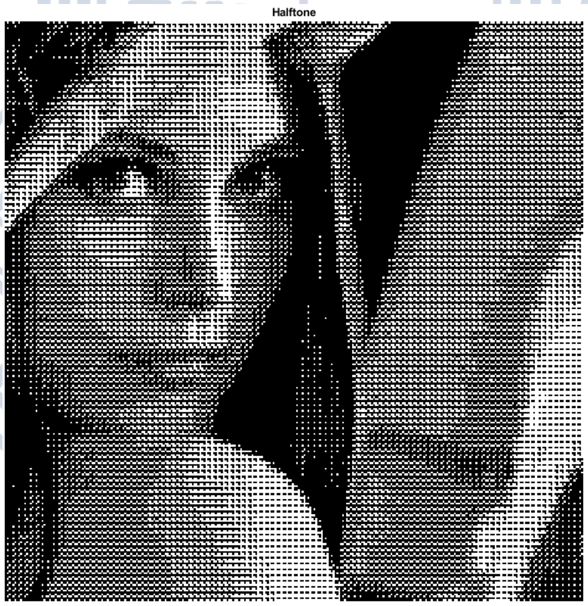


Figure 16: Halftone face image

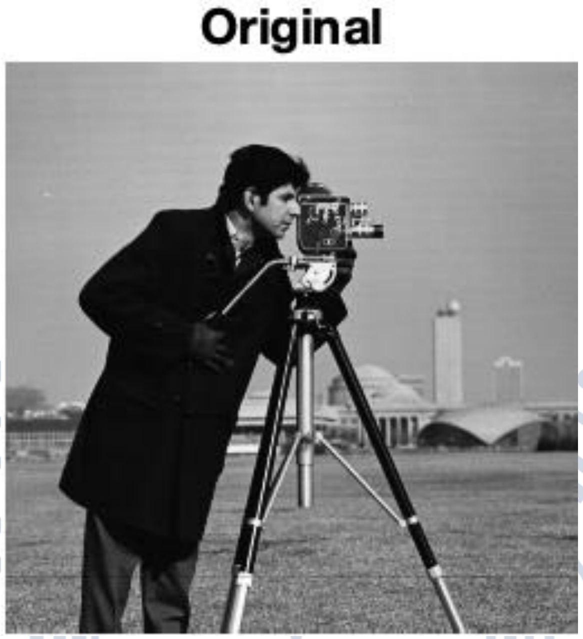


FIGURE 17: ORIGINAL CAMERAMAN IMAGE

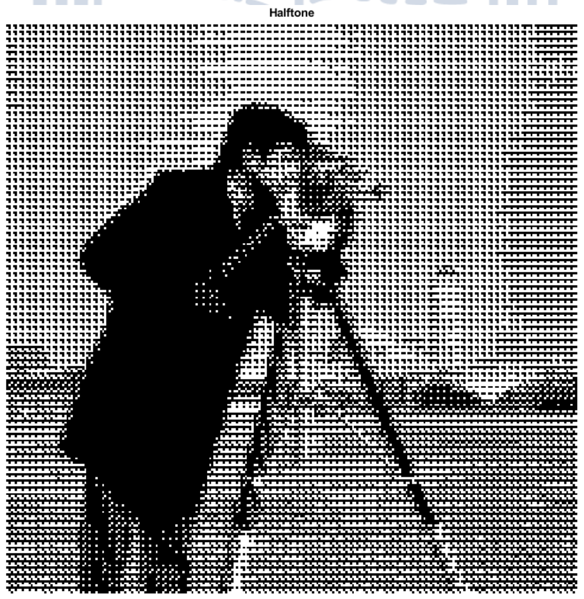


FIGURE 18: HALFTONE CAMERAMAN IMAGE

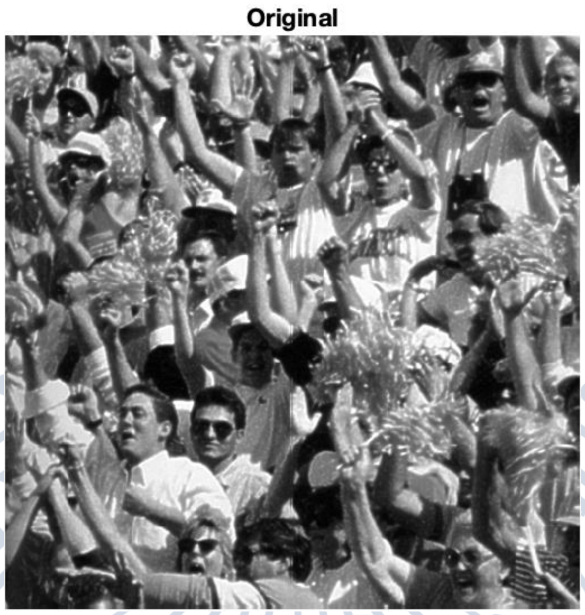


FIGURE 19: ORIGINAL CROWD IMAGE

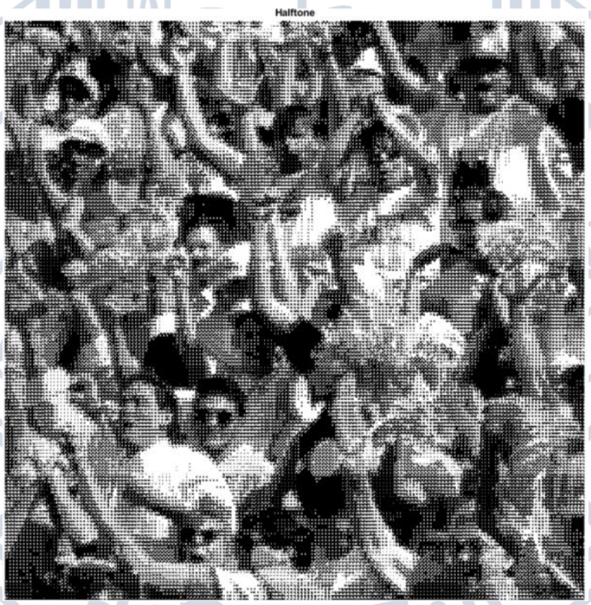


FIGURE 20: HALFTONE CROWD IMAGE

Upon closer review of each image, it is evident that the halftone transformation was successful.

Dynamic Image Sizes and Edge Correction

This script works for images of varying dimensions, including heights and widths that are now evenly divisible by groups of three. The halftone function accommodates for this potential scenario and performs halftone transformation along edges of the input image.
In Figure 21, a zoomed-up region of the cameraman image in its lower-right corner shows that the halftone transformation, and that its right and bottom edges have been accounted for. Its lower-right edge was also considered, and the halftone image shows a successful transformation.

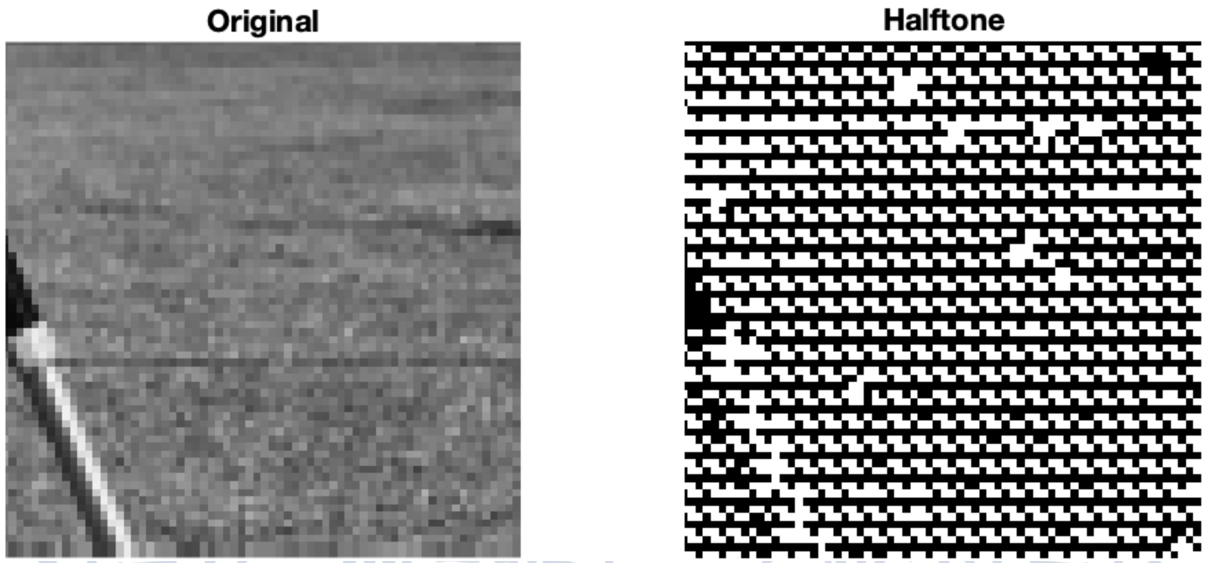


Figure 21: Original and halftone cameraman images are compared side-by-side to show successful transformation

With the side-by-side comparison, it is evident that the halftone transformation was successful. In Figure 22, another side-by-side comparison is made in the third image.

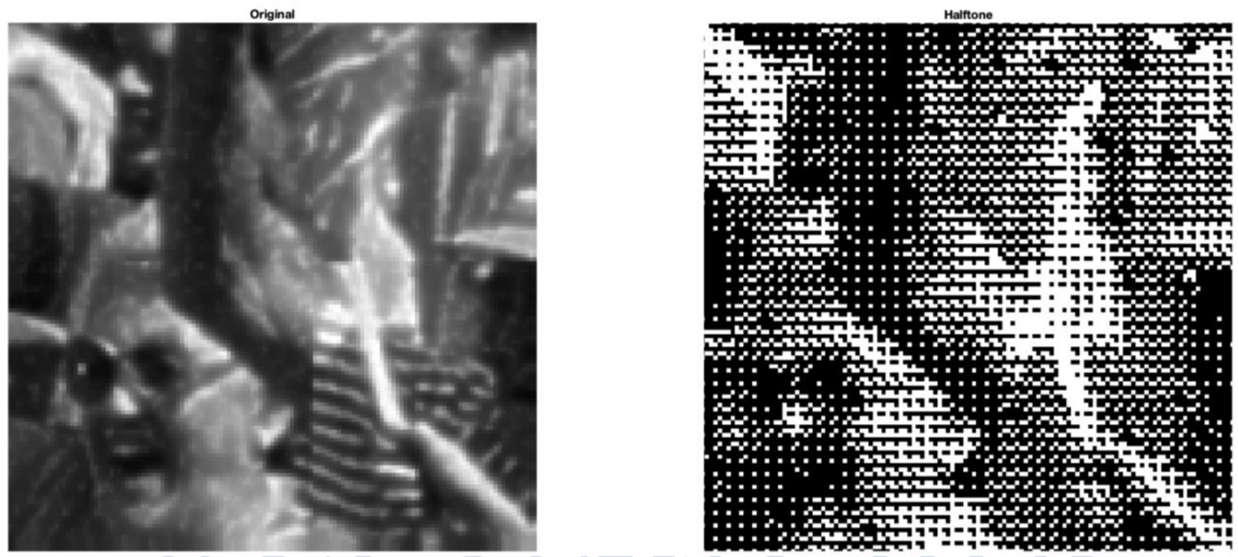


Figure 22: Original and halftone crowd images are compared side-by-side to show successful transformation

Creating a Wedge to Prove Halftone Function Effectiveness

Figure 23 contains a grayscale image of size `256x256` and contains a gradient going from top to bottom from dark to light pixels. The value of the pixels at the top of the gradient image is `0`, and through each row, the value of each pixel increments by `1`. At the end of the wedge (bottom edge), the pixel values are `255`, the maximum value a pixel can be in a grayscale image. The script to create a grayscale image and assigning pixel values to each row in the image is shown in Figure 24.

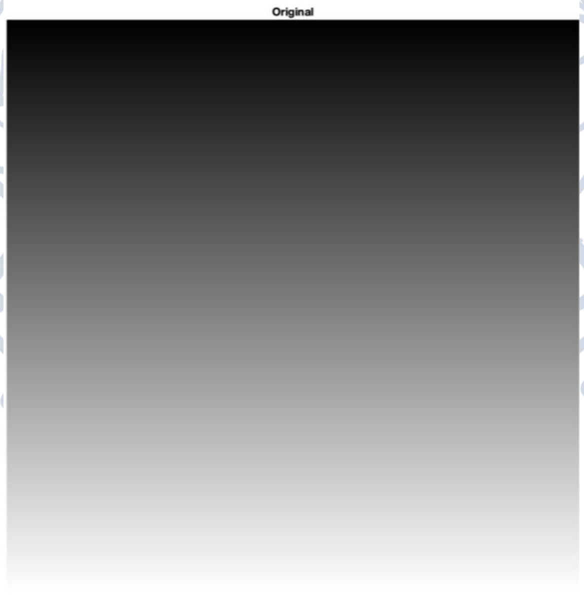


Figure 23: Grayscale gradient image of 256x256 pixels

```
%{
Write a test script that generates a test pattern image consisting of
a grey scale "wedge" of size 256x256, whose first row is all 0, the next
row is all 1, and so on, with the last row being 255.
%}

temp = zeros(256,256);
inputWedge = im2uint8(temp);
for rows = 1:size(inputWedge,1)
    for cols = 1:size(inputWedge,2)
        inputWedge(rows,cols) = rows-1;
    end
end
```

FIGURE 24: SCRIPT CREATES WEDGE IMAGE IN GRAYSCALE

The grayscale image is then sent through the halftone process by using its variable `inputWedge` to call the `halftone` function (Figure 25) and the image is prepared. The resulting image can be seen in Figure 26.

```
outputWedge = halftone(inputWedge);
figure()
imshow(inputWedge);
title("Original");
pause(0.3)
figure()
imshow(outputWedge);
title("Halftone");
pause(0.3)
```

FIGURE 25: CALLING THE HALFTONE FUNCTION USING THE GENERATED GRAYSCALE IMAGE

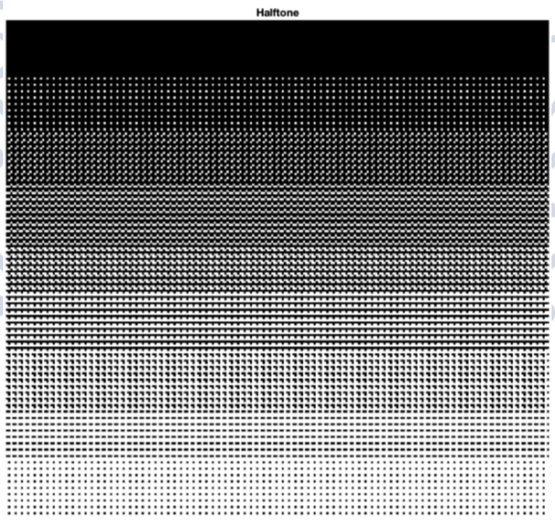


FIGURE 26: GRAYSCALE GRADIENT IMAGE OF 256X256 PIXELS AFTER HALFTONE RENDERING

Using the halftone transformed image, it is possible to confirm that the transformation was successful. Upon closer inspection (Figure 27), it is evident that the transformation was successful at rendering the edges (right, bottom, and lower-right corner) – considering that `256x256` is not evenly divisible by three.

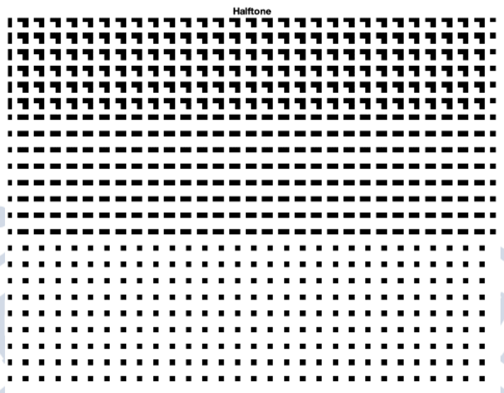


FIGURE 27: CLOSE UP OF THE BOTTOM-RIGHT CORNER OF THE WEDGE IMAGE

In conclusion, the halftone transformation worked successfully, and the wedge image was generated and transformed appropriately.

CONCLUSION

After reviewing the previous images produced by the halftone function, it is apparent that the transformations and rendering processes were successful. Looking at the results, the halftone images look very similar to the original grayscale images. The bitmap arrays that were used to assign grayscale pixel regions to black or white proved to be usable and viable options for this rendering technique. Additionally, the function is usable by different image sizes, which suggests that any image could be inserted into this function and a halftone render could be produced. Since the algorithms used in the halftone function accommodate for edges, the program should not fail, and it operates efficiently. By using a linear interpolation method and dividing the `255` into ten different ranges, it is possible to use all bitmap arrays to interpolate different halftone regions in the input image to produce a concise output image. All the images were successfully transformed, and all images kept their same dimensions, therefore the function works as expected without modifying files or doing any pixel count manipulation.

APPENDIX A

THE FOLLOWING INFORMATION INCLUDES SOURCES USED TO GENERATE SCRIPTS AND BUILD FUNCTIONS IN `HALFTONE_GENERATOR.M`. FOR MORE INFORMATION OF A SPECIFIC TOPIC, REFER TO THE LINK PROVIDED.

---

[1] Creating greyscale image
MathWorks, (2022). "How to create a gray scale image". Retrieved Dec 2022 from
https://www.mathworks.com/matlabcentral/answers/108006-how-to-create-a-gray-scale-image

[2] Converting image to type uint8
MathWorks, (2022). im2uint8 (R2021b). Retrieved Dec 2022 from
https://www.mathworks.com/help/images/ref/im2uint8.html

[3] Remainder
MathWorks (2022). rem (R2021b). Retrieved Dec 2022 from https://www.mathworks.com/help/matlab/ref/rem.html

[4] "Ones" matrix generation
MathWorks, (2022). ones (R2021b). Retrieved Dec 2022 from
https://www.mathworks.com/help/matlab/ref/ones.html

[5] Accessing specific column range in an array
MathWorks (2022). "Specific column range from an array". Retrieved Dec 2022 from
https://www.mathworks.com/matlabcentral/answers/333727-specific-column-range-from-an-array

[6] Determining sum of elements in a matrix
MathWorks (2022). sum (R2021b). Retrieved Dec 2022 from
https://www.mathworks.com/help/matlab/ref/sum.html

[7] Using fprintf in MATLAB
MathWorks (2022). fprintf (R2021b). Retrieved Dec 2022 https://www.mathworks.com/help/matlab/ref/fprintf.html

[8] Rounding values in MATLAB
MathWorks (2022). round (R2021b). Retrieved Dec 2022 https://www.mathworks.com/help/matlab/ref/round.html

[9] Accessing the size of an array in MATLAB
MathWorks (2022). size (R2021b). Retrieved Dec 2022 https://www.mathworks.com/help/matlab/ref/size.html

[10] Taking a mean of all elements in an array
MathWorks (2022). mean (R2021b). Retrieved Dec 2022 https://www.mathworks.com/help/matlab/ref/mean.html

[11] Calculating remainder
MathWorks (2022). rem (R2021b). Retrieved Dec 2022 https://www.mathworks.com/help/matlab/ref/rem.html

[12] Binarize an image from grayscale
MathWorks (2022). imbinarize (R2021b). Retrieved Dec 2022
https://www.mathworks.com/help/images/ref/imbinarize.html

[13] Using progress bar (waitbar)
MathWorks (2022). waitbar (R2021b). Retrieved Dec 2022
https://www.mathworks.com/help/matlab/ref/waitbar.html

[14] Special characters in waitbar

MathWorks (2022). “How to write percentage sign into a txt file”. Retrieved Dec 2022
https://www.mathworks.com/matlabcentral/answers/376325-how-to-write-percentage-sign-into-a-txt-file

[15] Working with subplots
MathWorks (2022). Display Multiple Images (R2021b). Retrieved Dec 2022
https://www.mathworks.com/help/images/display-multiple-images.html

END OF APPENDIX A

APPENDIX B

Below is the script used to generate all relative figures and contains the halftone function.

BEGIN SCRIPT:

```
close all;
clear all;
%{
Creating greyscale image
https://www.mathworks.com/matlabcentral/answers/108006-how-to-create-a-gray-scale-
image

Converting image to type uint8
https://www.mathworks.com/help/images/ref/im2uint8.html

Remainder
https://www.mathworks.com/help/matlab/ref/rem.html

Ones matrix generation
https://www.mathworks.com/help/matlab/ref/ones.html

Accessing specific column range in an array
https://www.mathworks.com/matlabcentral/answers/333727-specific-column-range-from-an-
array

Determining sum of elements in a matrix
https://www.mathworks.com/help/matlab/ref/sum.html

Using fprintf
https://www.mathworks.com/help/matlab/ref/fprintf.html

Rounding values in MATLAB
https://www.mathworks.com/help/matlab/ref/round.html

Accessing the size of an array in MATLAB
https://www.mathworks.com/help/matlab/ref/size.html

Taking a mean of all elements in an array
https://www.mathworks.com/help/matlab/ref/mean.html

Calculating remainder
https://www.mathworks.com/help/matlab/ref/rem.html

Binarize an image from grayscale
https://www.mathworks.com/help/images/ref/imbinarize.html

Using progress bar (waitbar)
https://www.mathworks.com/help/matlab/ref/waitbar.html

Special characters in waitbar
https://www.mathworks.com/matlabcentral/answers/376325-how-to-write-percentage-sign-
into-a-txt-file
Working with subplots
https://www.mathworks.com/help/images/display-multiple-images.html


%}
% May want to consider linear interpolation
% floor (average/range of pixels) * 10

inputMatrixA = im2uint8(imread("Fig0225(a)(face).tif"));
inputMatrixB = im2uint8(imread("Fig0225(b)(cameraman).tif"));
inputMatrixC = im2uint8(imread("Fig0225(c)(crowd).tif"));
% inputMatrixX = im2uint8(imread("test4.jpg")); % TEST IMAGE
% inputMatrixX2 = im2uint8(imread("test5.jpg")); % TEST IMAGE

%{
Write a test script that generates a test pattern image consisting of
a grey scale "wedge" of size 256x256, whose first row is all 0, the next
row is all 1, and so on, with the last row being 255.
%}

temp = zeros(256,256);
inputWedge = im2uint8(temp);
for rows = 1:size(inputWedge,1)
    for cols = 1:size(inputWedge,2)
        inputWedge(rows,cols) = rows-1;
    end
end

%% MAKE A WEDGE OF ANY DIMENSIONS GREATER THAN 255 x 255 PIXELS
% stop = false;
% while(true)
%     x = input("Enter the height: ");
%     if (x > 255)
%         break;
%     elseif (x == -1)
%         stop = true;
%         break;
%     else
%         fprintf("Invalid size! Enter number greater than 255 or -1 to cancel!\n");
%     end
% end
% if (stop~=true)
%     while (true)
%         y = input("Enter the width: ");
%         if (y > 255)
%             break;
%         elseif (y == -1)
%             stop = true;
%             break;
%         else
%             fprintf("Invalid size! Enter number greater than 255 or -1 to
cancel!\n");
%         end
%     end
% end
%
% if (stop == true)
%     fprintf("Process terminated!\n\n")
% else
%     temp = ones(x,y);
%     ver = round(size(temp,2)/256);
%     matrix = im2uint8(temp);
%     for rows = 1:size(matrix,1)
%         i = 0;
%         for cols = 1:ver:size(matrix,2)
%             if cols+ver>size(matrix,2)
%                 break;
%             else
%                 matrix(rows,cols:(cols+ver)) = i;
%                 i = i + 1;
%             end
%         end
%     end
%     newWedge = halftone(matrix);
%     figure()
%     subplot(1,2,1), imshow(matrix);
%     title("Your Original");
%     subplot(1,2,2), imshow(newWedge);
%     title("Halftone");
% end
%%

outputMatrixA = halftone(inputMatrixA);
figure()
imshow(imread("Fig0225(a)(face).tif"));
title("Original");
pause(0.3)
figure()
imshow(outputMatrixA);
title("Halftone");
pause(0.3)

outputMatrixB = halftone(inputMatrixB);
figure()
imshow(imread("Fig0225(b)(cameraman).tif"));
title("Original");
pause(0.3)
figure()
imshow(outputMatrixB);
title("Halftone");
pause(0.3)

outputMatrixC = halftone(inputMatrixC);
figure()
imshow(imread("Fig0225(c)(crowd).tif"));
title("Original");
pause(0.3)
figure()
imshow(outputMatrixC);
```

```
title("Halftone");
pause(0.3)

outputWedge = halftone(inputWedge);
figure()
imshow(inputWedge);
title("Original");
pause(0.3)
figure()
imshow(outputWedge);
title("Halftone");
pause(0.3)

% INSERT A TEST IMAGE
% outputMatrixX = halftone(rgb2gray(inputMatrixX)); % TEST IMAGE
% figure()
% subplot(1,2,1), imshow(imread("test.jpg"));
% title("Original");
% subplot(1,2,2), imshow(outputMatrixX);
% title("Halftone");


%% FUNCTION SCRIPT
function output = halftone(inputImage)
    %{
    halftone  Converts a grayscale image to a binary image by using binary dot
patterns to
    render grayscale values.
    Syntax:
        out = halftone(in)

    Input:
        in = the grayscale image to be rendered. It should be of type uint8 and have
values in the range 0-255.

    Output:
        out = the rendered binary image. It is of type uint8 and will have two
values: 0 and 255.

    History:
        See GitHub commit history:
        https://github.com/alejandrohsanchez/Digital-Image-Processing-
Projects/commits/main
    %}
    A = inputImage;
    % Number of pixel rows and columns in the image
    rows = size(A,1);
    cols = size(A,2);
    %Detecting if image is not divisible by 3 evenly
    r_remain = rem(rows, 3);
    c_remain = rem(cols, 3);

    % Creating halftone transform matrices
    % 0 is black, 255 is white
    dot0 = [0 0 0; 0 0 0; 0 0 0];
    dot1 = [0 255 0; 0 0 0; 0 0 0];
    dot2 = [0 255 0; 0 0 0; 0 0 255];
    dot3 = [255 255 0; 0 0 0; 0 0 255];
    dot4 = [255 255 0; 0 0 0; 255 0 255];
    dot5 = [255 255 255; 0 0 0; 255 0 255];
    dot6 = [255 255 255; 0 0 255; 255 0 255];
    dot7 = [255 255 255; 0 0 255; 255 255 255];
    dot8 = [255 255 255; 255 0 255; 255 255 255];
    dot9 = [255 255 255; 255 255 255; 255 255 255];

    f = waitbar(100, "Media Read!", "Name", "Halftone Transformation Progress",
"CreateCancelBtn",...
        "setappdata(gcbf,'canceling',1)");
    pause(0.3)
    kill = false;
    progress = 0;
    setappdata(f,"canceling",0);
    % UPPER-LEFT LEFT-TO-RIGHT TRANSFORM
    localTicks = (rows-r_remain) * (cols-c_remain);
    currentTick = 0;
    for row_idx = 1:3:rows-r_remain
        if (c_remain > 0 && r_remain > 0)
            progress = (row_idx/(rows-r_remain))*0.25;
        elseif ((c_remain > 0 && r_remain==0) || (c_remain==0 && r_remain>0))
            progress = (row_idx/(rows-r_remain))*0.33;
        else
            progress = (row_idx/(rows-r_remain));
        end
        waitbar(progress,f,sprintf("Processing Halftone Transformation %.f%%",
(currentTick/localTicks)*100));
        if getappdata(f,"canceling")
            kill = true;
            break
        end
        for col_idx = 1:3:cols-c_remain
            if (col_idx+2<=cols)
                currentTick = currentTick + 9;
                % Normal preparation for transform
                PXL_AVG = round(mean(A(row_idx:row_idx+2,col_idx:col_idx+2), "all"));
                if (PXL_AVG>=0 && PXL_AVG<=25)
                    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot0;
                elseif (PXL_AVG>=26 && PXL_AVG<=51)
                    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot1;
                elseif (PXL_AVG>=52 && PXL_AVG<=77)
                    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot2;
                elseif (PXL_AVG>=78 && PXL_AVG<=103)
                    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot3;
                elseif (PXL_AVG>=104 && PXL_AVG<=129)
                    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot4;
                elseif (PXL_AVG>=130 && PXL_AVG<=155)
                    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot5;
                elseif (PXL_AVG>=156 && PXL_AVG<=181)
                    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot6;
                elseif (PXL_AVG>=182 && PXL_AVG<=207)
                    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot7;
                elseif (PXL_AVG>=208 && PXL_AVG<=233)
                    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot8;
                elseif (PXL_AVG>=234 && PXL_AVG<=255)
                    A(row_idx:row_idx+2,col_idx:col_idx+2) = dot9;
                end
            else
                break
            end
        end
        if (row_idx+3>=rows)
            break;
        end
    end
    waitbar(progress,f,sprintf("Processing Halftone Transformation %.f%%",
(currentTick/localTicks)*100));

    % PROCESSING EDGES %
    % RIGHT EDGE TRANSFORM
    if ((r_remain>0 || c_remain>0) && getappdata(f,"canceling")==false)
        localTicks = ((rows-r_remain) * c_remain) + ((cols-c_remain) * r_remain) +
(r_remain * c_remain);
        currentTick = 0;
    end
    if (c_remain > 0 && kill==false)
        col_idx = cols-c_remain+1;
        for row_idx = 1:3:rows
            if getappdata(f,"canceling")
                kill = true;
                break
            end
            if (row_idx+2<=rows)
                if (c_remain > 0 && r_remain > 0)
                    progress = ((row_idx/(rows-r_remain))*0.25)+0.25;
                elseif (c_remain > 0 && r_remain==0)
                    progress = ((row_idx/(rows-r_remain))*0.33)+0.33;
                end
                waitbar(progress,f,sprintf("Cleaning Right Edge %.f%%",
(currentTick/localTicks)*100));
                currentTick = currentTick + (3 * c_remain);
                PXL_AVG = round(mean(A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1),
"all"));
                if (PXL_AVG>=0 && PXL_AVG<=25)
                    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) =
dot0(:,1:c_remain);
                elseif (PXL_AVG>=26 && PXL_AVG<=51)
                    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) =
dot1(:,1:c_remain);
                elseif (PXL_AVG>=52 && PXL_AVG<=77)
                    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) =
dot2(:,1:c_remain);
                elseif (PXL_AVG>=78 && PXL_AVG<=103)
                    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) =
dot3(:,1:c_remain);
                elseif (PXL_AVG>=104 && PXL_AVG<=129)
```

```
                    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) =
dot4(:,1:c_remain);
                elseif (PXL_AVG>=130 && PXL_AVG<=155)
                    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) =
dot5(:,1:c_remain);
                elseif (PXL_AVG>=156 && PXL_AVG<=181)
                    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) =
dot6(:,1:c_remain);
                elseif (PXL_AVG>=182 && PXL_AVG<=207)
                    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) =
dot7(:,1:c_remain);
                elseif (PXL_AVG>=208 && PXL_AVG<=233)
                    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) =
dot8(:,1:c_remain);
                elseif (PXL_AVG>=234 && PXL_AVG<=255)
                    A(row_idx:row_idx+2,col_idx:col_idx+c_remain-1) =
dot9(:,1:c_remain);
                end
            end
        end
        waitbar(progress,f,sprintf("Cleaning Right Edge %.f%%",
(currentTick/localTicks)*100));
    end

    % BOTTOM EDGE TRANSFORM
    if (r_remain > 0 && kill==false)
        row_idx = rows-r_remain+1;
        for col_idx = 1:3:cols
            if getappdata(f,"canceling")
                kill = true;
                break
            end
            if (col_idx+2<=cols)
                if (c_remain > 0 && r_remain > 0)
                    progress = ((col_idx/(cols-c_remain))*0.25)+0.50;
                elseif (c_remain==0 && r_remain>0)
                    progress = ((col_idx/(cols-c_remain))*0.33)+0.33;
                end
                waitbar(progress,f,sprintf("Cleaning Bottom Edge %.f%%",
(currentTick/localTicks)*100));
                currentTick = currentTick + (3 * r_remain);
                PXL_AVG = round(mean(A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2),
"all"));
                if (PXL_AVG>=0 && PXL_AVG<=25)
                    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) =
dot0(1:r_remain,:);
                elseif (PXL_AVG>=26 && PXL_AVG<=51)
                    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) =
dot1(1:r_remain,:);
                elseif (PXL_AVG>=52 && PXL_AVG<=77)
                    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) =
dot2(1:r_remain,:);
                elseif (PXL_AVG>=78 && PXL_AVG<=103)
                    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) =
dot3(1:r_remain,:);
```

```
                elseif (PXL_AVG>=104 && PXL_AVG<=129)
                    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) =
dot4(1:r_remain,:);
                elseif (PXL_AVG>=130 && PXL_AVG<=155)
                    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) =
dot5(1:r_remain,:);
                elseif (PXL_AVG>=156 && PXL_AVG<=181)
                    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) =
dot6(1:r_remain,:);
                elseif (PXL_AVG>=182 && PXL_AVG<=207)
                    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) =
dot7(1:r_remain,:);
                elseif (PXL_AVG>=208 && PXL_AVG<=233)
                    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) =
dot8(1:r_remain,:);
                elseif (PXL_AVG>=234 && PXL_AVG<=255)
                    A(row_idx:row_idx+r_remain-1,col_idx:col_idx+2) =
dot9(1:r_remain,:);
                end
            end
        end
        waitbar(progress,f,sprintf("Cleaning Bottom Edge %.f%%",
(currentTick/localTicks)*100));
    end

    % BOTTOM-RIGHT CORNER TRANSFORM
    if ((r_remain > 0 && kill==false) || (c_remain > 0 && kill==false))
        row_idx = rows-r_remain+1;
        col_idx = cols-c_remain+1;
        if (c_remain > 0 && r_remain > 0)
            progress = 0.75;
        elseif ((c_remain > 0 && r_remain==0) || (c_remain==0 && r_remain>0))
            progress = 0.67;
        end
        waitbar(progress,f,sprintf("Touching Up Corners %.f%%",
(currentTick/localTicks)*100));
        currentTick = currentTick + (r_remain * c_remain);
        progress = 1;
        waitbar(progress,f,sprintf("Touching Up Corners %.f%%",
(currentTick/localTicks)*100));
        PXL_AVG = round(mean(A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-
1), "all"));
        if (PXL_AVG>=0 && PXL_AVG<=25)
            A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) =
dot0(1:r_remain,1:c_remain);
        elseif (PXL_AVG>=26 && PXL_AVG<=51)
            A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) =
dot1(1:r_remain,1:c_remain);
        elseif (PXL_AVG>=52 && PXL_AVG<=77)
            A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) =
dot2(1:r_remain,1:c_remain);
        elseif (PXL_AVG>=78 && PXL_AVG<=103)
            A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) =
dot3(1:r_remain,1:c_remain);
        elseif (PXL_AVG>=104 && PXL_AVG<=129)
```

```
            A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) =
dot4(1:r_remain,1:c_remain);
        elseif (PXL_AVG>=130 && PXL_AVG<=155)
            A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) =
dot5(1:r_remain,1:c_remain);
        elseif (PXL_AVG>=156 && PXL_AVG<=181)
            A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) =
dot6(1:r_remain,1:c_remain);
        elseif (PXL_AVG>=182 && PXL_AVG<=207)
            A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) =
dot7(1:r_remain,1:c_remain);
        elseif (PXL_AVG>=208 && PXL_AVG<=233)
            A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) =
dot8(1:r_remain,1:c_remain);
        elseif (PXL_AVG>=234 && PXL_AVG<=255)
            A(row_idx:row_idx+r_remain-1,col_idx:col_idx+c_remain-1) =
dot9(1:r_remain,1:c_remain);
        end
    end
    if (kill==false)
        pause(0.3);
        output = A;
        delete(f)
    else
        waitbar(progress,f,sprintf("Processing Interrupted! %.f%%",
(currentTick/localTicks)*100));
        pause(0.3);
        delete(f);
        output = A;
    end
end
```

END OF SCRIPT

END OF APPENDIX B