# 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.

# 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.

A picture containing text

Description automatically generated

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.

Text

Description automatically generated

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. These dimensions will be useful later when the function performs the halftoning calculations.

Text

Description automatically generated

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.

Text

Description automatically generated

Figure 4: Arrays 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.

Chart, shape, arrow, bubble chart

Description automatically generated

Figure 5: Raster images of black (255 grayscale value) and white (0 grayscale value) pixels