Lab 02

Basic Image Processing
Fall 2022

First part: Dithering

You should remember…

Dithering is a technique in which you add noise to the image in order to preserve some kind of spatial information during quantization. (Although due to the additive noise the quality of certain image parts (e.g. edges) will decrease).

Image source: https://hu.wikipedia.org/wiki/Dithering

Dithering with white noise

The algorithm is the following:

`for` every pixel of the image:

Generate a random number `r` from the `[0, Fmax]` uniform distribution

Add this value to the grayscale intensity of the current pixel: `p = p + r`

Threshold the pixel value using `Fmax` as threshold

Ordered dithering – general idea

We create a new image by replicating a *threshold map*. This new image has the same size as the image we want to quantize.

The threshold map contains threshold levels. If the pixel intensity in the original image is higher than the threshold value then the pixel becomes white in the output image; otherwise it will be black.

The clever arrangement of the threshold levels in the map yields good results.

Ordered dithering – threshold maps

For this kind of dithering we use a so called Threshold map (which is a matrix).

The recursive rule of such a matrix-series is the following:

$$D_0 = \begin{bmatrix} 0 \end{bmatrix}$$

$$D_{n+1} = \begin{bmatrix} 4\,D_n + 0 & 4\,D_n + 2 \\ 4\,D_n + 3 & 4\,D_n + 1 \end{bmatrix}$$

The normalization factor for the n-th matrix is $\frac{1}{4^n}$.

The n-th normalized threshold map is $\bar{D}_n = \frac{D_n}{4^n}$

Now please

download the ‘Lab 02’ code package

from the

moodle system

Exercise 1

Implement the function `random_dither` in which:

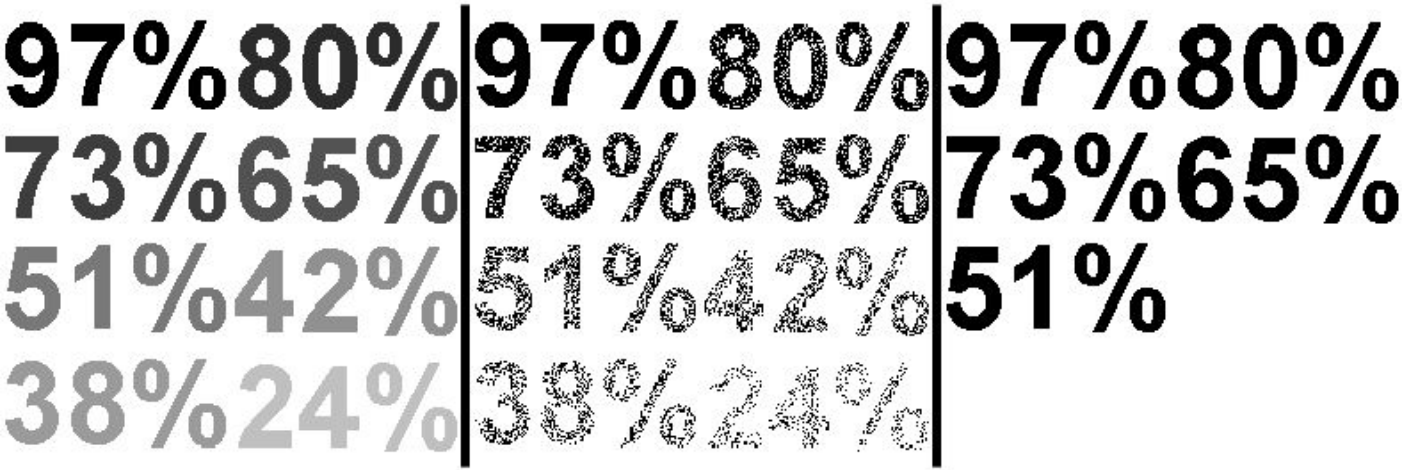
- Generate a matrix of random values between 0 and 1. The matrix should have the same size as the input of the function. Use `rand()`.
- Add the values of the random matrix to the input image.
- Threshold the newly created image.

You can assume that the input of the function is a double type grayscale image with values in the [0,1] range.

Run `script1.m` and examine the results.

Dithering with white noise

Original grayscale image | Dithering with white noise | Threshold at 50%



Exercise 2

Implement the function `ord_dit_matrix` in which:

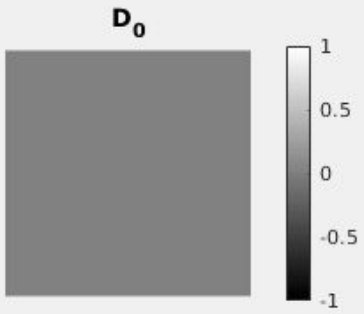
- Using the recursive rule mentioned in Slide 5 the function should be able to generate the appropriate normalized threshold maps.
  - This recursive rule was

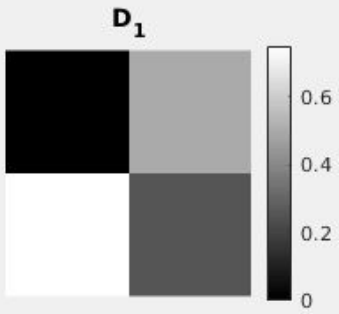
$$D_0 = \begin{bmatrix} 0 \end{bmatrix}$$

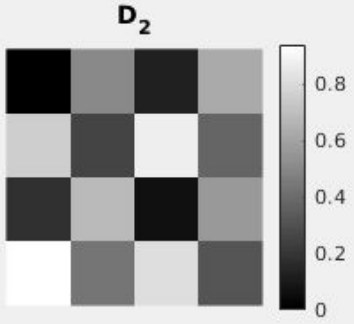
$$D_{n+1} = \begin{bmatrix} 4\,D_n + 0 & 4\,D_n + 2 \\ 4\,D_n + 3 & 4\,D_n + 1 \end{bmatrix}$$

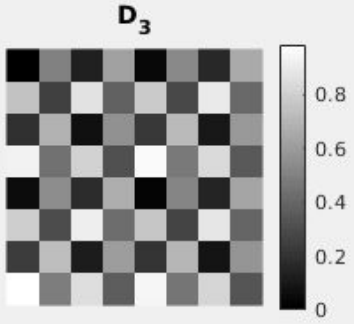
And the normalization factor is $\frac{1}{4^n}$

The function should return a matrix in which all the values are in the [0,1] range.
Run `script2.m` and examine the results.


D_0
1
0.5
0
-0.5
-1


D_1
0.6
0.4
0.2
0


D_2
0.8
0.6
0.4
0.2
0


D_3
0.8
0.6
0.4
0.2
0

Exercise 3

Implement the function `ordered_dither` in which:

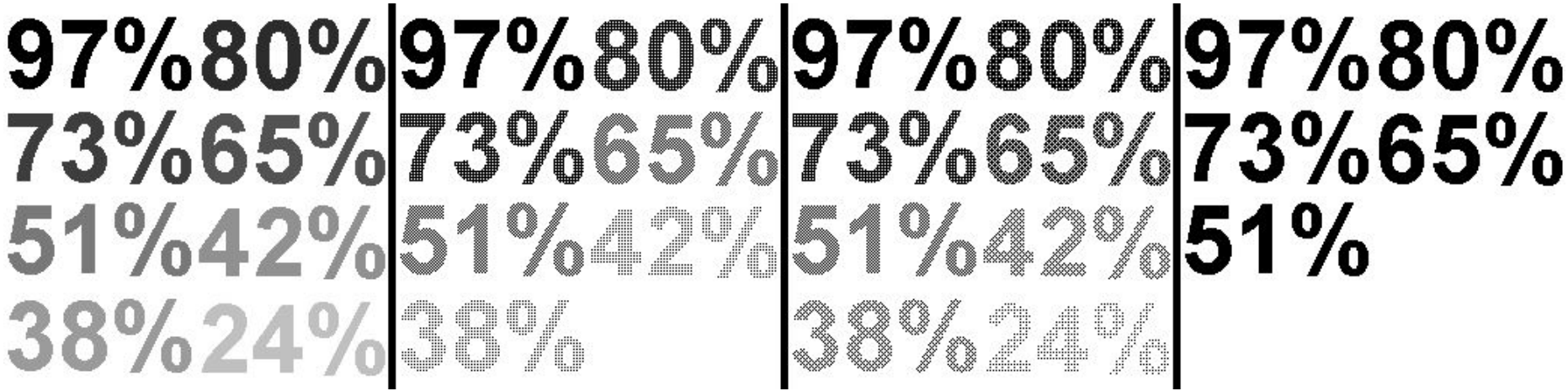
- Get the size of the input image (`I`)
- Get the size of the threshold map (`D`)
- Using `repmat()` replicate `D` to create a matrix which has the same size as `I`.
- Threshold the image `I` with this newly created matrix and return the result.

The function should return an image in which all the values are either 0 or 1. You can assume that the input image `I` is always double-type grayscale image (values in range [0,1]) and the size of this image is an integral multiple of the size of `D`.

Run `script3.m` and examine the results.

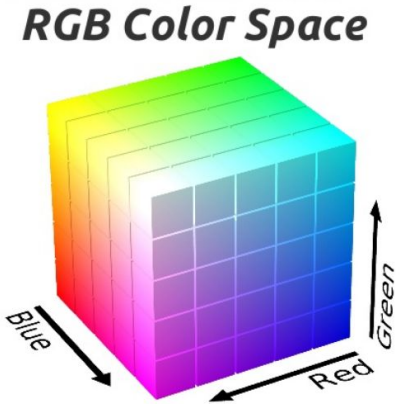
Ordered dithering

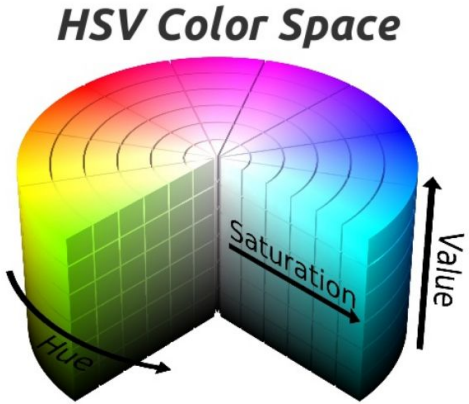
Original grayscale image | Dithering with D1 | Dithering with D2 | Threshold at 50%



Second part: Color spaces

You should remember…





`rgb2hsv()`	
RGB →	HSV
`uint8` ([0, 255]) `double` ([0, 1]) →	`double` ([0, 1])

`hsv2rgb()`	
HSV →	RGB
`double` ([0, 1]) →	`double` ([0, 1])

Image source: https://www.slideshare.net/michelalves/about-perception-and-hue-histograms-in-hsv-space

Thresholding, segmentation

- Thresholding / binarization with a single number:
  If the pixel intensity in the original image is higher than the threshold value then the pixel becomes white in the output image; otherwise it will be black.

```
R = squeeze(I(:, :, 1));
R_t = R > 128;
```

selecting the red-channel
bonus: if the singleton is the last dim, `squeeze` is not necessary

binarization at mid-gray

- Segmentation:
  The process of partitioning an image into multiple segments (sets of pixels, also known as image objects).

(images from: https://towardsdatascience.com/semantic-segmentation-of-150-classes-of-objects-with-5-lines-of-code-7f244fa96b6c)

Exercise 4

Implement the function `find_the_duck()` in which:

- Segment the input image based on color channels. You can use color space transformations and thresholding only.
- The function should return a logical matrix where true values indicate ‘duck’.
- It is required to have an **error value less than 1.5%**

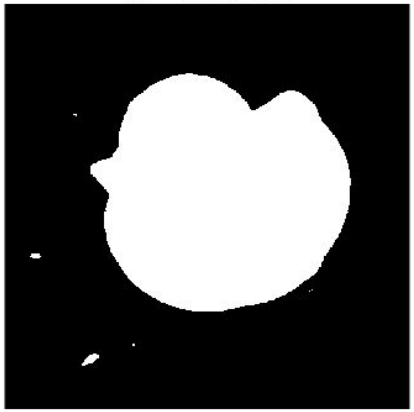
Implement the function `find_the_pine()` in which:

- Segment the input image based on color channels. You can use color space transformations and thresholding only.
- The function should return a logical matrix where true values indicate ‘pine’.
- It is required to have an **error value less than 3.0%**

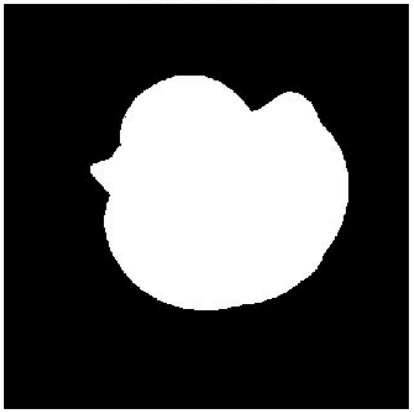
Run `script4.m` and examine the results.

Original image

Your segmentation
err = 0.63362%



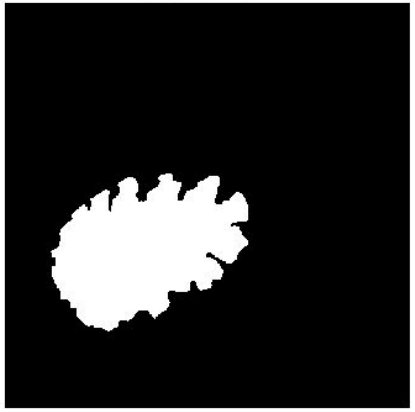
Ground truth



Original image

Your segmentation
err = 2.6821%

Ground truth



THE END