Project 1 - Halftoning

Thales Mateus Rodrigues Oliveira (RA 148051)

***Abstract*—**

I. Introduction

Halftoning is a technique which is used for reducing the number of colors used to represent an image, while is desired to keep a good visual perception of its contents for the user. In this work, the implementation of halftoning techniques with error diffusion was realized. For each technique, tests were executed to analyze the quality of the output image, sweeping the image in two different ways. The next section explains the implemented algorithms and the following, the tests done and the discussion.

The code, along with the input files and the report is delivered in the compressed file THALES_MATEUS_RODRIGUES_OLIVEIRA_148051.tar, in the Google Classroom.

II. The Program

The program was implemented with Python 3.7.3. The libraries used and their respective versions are OpenCV 4.1.0 and Numpy 1.16.4.

A. How to execute it

The project has a Makefile available to help performing some actions on it. The Makefile has 3 basic instructions: clean, build and exec. Clean instruction removes generated images stored in the **output** folder, the source code in **bin** folder and the folders itself. The Build instruction creates the **output** and **bin** folders, and move execution code to **bin**. The Exec instruction executes the code with images in the **input** folder. Listing 1 provides examples of how to execute the three instructions in a terminal.

```
#clean environment, deletes output and bin folders
   and their content
make clean

#prepare the environment for code execution
make build

#executes code
make exec
```

Listing 1. Makefile usage example

B. Input

The program does not have an input argument by default, the input images are listed in code, and they are expected to be stored in the **input** folder. Listing 2 shows how images are listed to be executed in code. The $images$ tuple is implemented in $src/main.py$

```
# for inserting other images, add tem to /input
  folder and list them here
images = (
    'baboon',
    'monalisa',
    'peppers',
    'watch'
)
```

Listing 2. Input images inside code

C. Output

The output of the program is a series of halftoning images based on the input ones, changing the error diffusion methods and the sweeping directions. The output images are stored in the **output** folder, and the images are labeled by concatenating the image name, whether is colored or grayscale, the error diffusion method and the sweep order (*e.g.: baboon_colored_sierra_left-to-right.png*)

D. Implementation

The function which implements the halftoning operation is defined in the *src/halftoning.py* file. The file contains a dictionary that lists the approaches used for error diffusion, as mentioned in Figure 1 in the project proposal. Listing 3 shows the implemented dictionary.

```
# Masks used for error propagation
MASKS = {
    "floyd-steinberg": np.array([
      [0, 0, 7/16],
      [3/16, 5/16, 1/16]]),
    "stevenson-arce": np.array([
      [0, 0, 0, 0, 0, 32/200, 0],
      [12/200, 0, 26/200, 0, 30/200, 0, 16/200],
      [0, 12/200, 0, 26/200, 0, 12/200, 0],
      [5/200, 0, 12/200, 0, 12/200, 0, 5/200]]),
    "burkes": np.array([
      [0, 0, 0, 8/32, 4/32],
      [2/32, 4/32, 8/32, 4/32, 2/32]]),
    "sierra": np.array([
      [0, 0, 0, 5/32, 3/32],
      [2/32, 4/32, 5/32, 4/32, 2/32],
      [0, 2/32, 3/32, 2/32, 0]]),
    "stucki": np.array(
      [[0, 0, 0, 8/42, 4/42],
      [2/42, 4/42, 8/42, 4/42, 2/42],
      [1/42, 2/42, 4/42, 2/42, 1/42]]),
    "jarvis-judice-ninke": np.array([
      [0, 0, 0, 7/48, 5/48],
      [3/48, 5/48, 7/48, 5/48, 3/48],
      [1/48, 3/48, 5/48, 3/48, 1/48]])
}
```

Listing 3. Masks used for error diffusion

The function *apply_halftoning* implements the desired operation. It receives the original image, the name of the error diffusion approach, the sweep method (left to right or alternated), and a benchmarking flag to monitor execution time. Listing 4 shows keyparts of implementation

ra148051@students.ic.unicamp.br

```python
def apply_halftoning(img, err_method="floyd-steinberg", sweep_method=1, benchmarking=False):
  # initialize result array
  result = np.zeros_like(img)
  # separates the masks that could be used
  # (it needs the flip version of mask for alternated sweep)
  m = (0, MASKS[err_method], np.flip(MASKS[err_method], 1))
  # saves mask dimensions to be used when needed
  mask_h, mask_w = m[1].shape
  # saves image dimensions to be used when needed
  img_h, img_w = img.shape
  # it holds the offset of manipulated pixel related to mask
  offset = mask_w//2
  # applies padding to image to make it easier the operations with mask
  img_padded = np.pad(img, ((0, mask_h - 1), (mask_w//2, mask_w//2)), 'constant')

  # default starting direction (left to right)
  direction = 1
  # default index values for sweeping from left to right and right to left
  sweep_options = (0, (0, img_w), (img_w - 1, 0))

  # it sweeps the image from top to bottom
  for j in range(img_h):
      # it sweeps the image from left to right or right to left depending on direction
      beginning, end = sweep_options[direction]
      # do the horizontal sweeping
      for i in range(beginning, end, direction):
          # depending on analyzed pixel, set its result value according to threshold
          if img_padded[j][(i + offset)] < 128:
              result[j][i] = 0
          else:
              result[j][i] = 1

          # calculates associated error
          error = img_padded[j][(i + offset)] - result[j][i]*255
          # propagates error according to mask
          img_padded[j:j+mask_h, i:i+mask_w] = (img_padded[j:j+mask_h, i:i+mask_w]
                                                + (error*m[direction])).astype(np.uint8)
      # it changes from left to right to right to left (vice-versa) depending on the sweep method
      direction *= sweep_method

  # scale the result
  return result*255
```

Listing 4. Implementation of halftoning solution

The decisions made in the implementation are explained in the following part.

1) Dealing with sweeping directions: In this solution, it was implemented two approaches: sweeping every line from left to right, and alternating directions when changing from one line to the other. Figure 1 shows the approaches. If the second mentioned method is chosen, the error diffusion mask has to be mirroed in the vertical direction. In other to avoid *if-else* statements when deciding whether mask to use for the specific line (the regular or the mirroed one), the tuple *m* holds both masks, and decide which one to use based on the *direction* flag (if *direction* is 1, use the regular mask. If it is -1, use the mirroed one). To sweep the image from right to left, it is also considered the indexes in the loop to decrease. Following the same idea of tuple *m*, the tuple *sweep_options* is used to set the starting index, the stop index, and the *direction* flag as step.

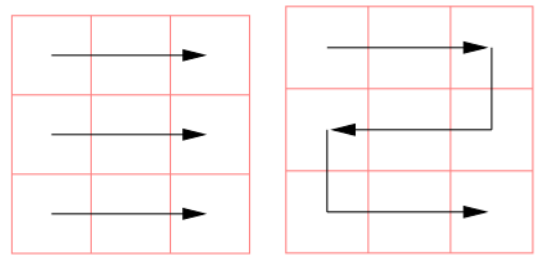

Fig. 1. Left: Sweeping from left to right. Right: Sweeping alternated

2) Image Padding to deal with borders: In order to deal with pixels in the border of the image, to diffuse the error, the image is padded as the mask propagates the error to its neighbors, as they may not exist for pixels in the border. The padding is done based on the mask properties, such as not diffusing errors to already visited pixels and the dimensions of each mask. The values padded does not matter, as they are not visited.

3) Defining the value of the pixel and the error diffusion: The main ideia of the halftoning algorithm is the following: rather than a simple binarization, in order to create the ”feeling” of having more than 2 intensities of a color in the image, an analyzed pixel propagates its information to specific close neighbors. This propagation of information is what we define as error diffusion. Based on a threshold which defines whether a pixel has the maximum or minimum intensity, the resulting pixel value is defined and the error is calculates and passed through specific neighbors with specific weights. That weight is what is defined in the error diffusion masks. The error diffused for the neighbor pixels is proportional to the mentioned weight. After performing these steps to all pixels in the image, the process is done. The steps are the ones inside the two *for* statements in listing 4. They were implemented based on Algorithm 5, presented on the *Enhancement* class of MO443: Introduction to Digital Image Processing course.

4) Sweeping the image: To perform the visit of every pixel, as seen in listing 4, it is used two nested *for* statements. As the author was unable to vectorize the error propagation, the execution time occours in the second scale. To define the value of the analyzed pixel in the iteration, an offset is used, as the pixel is in the middle column related to mask.

III. Experiments

The *src/main.py* file executes the test pipeline. The idea is the following: for each input image, for each error propagation method, for each sweep order, generate the grayscale and colored halftoning output images. The input images are stored in **input** folder as mentioned before, and their names and dimensions are listed in table I. The input images are shown in figure 2

Image Names	Dimensions (width x height)
baboon_colored.png	512 x 512
monalisa_colored.png	256 x 256
peppers_colored.png	512 x 512
watch_colored.png	Vertical 1024 x 768

TABLE I
INPUT IMAGES USED IN EXPERIMENTS

Fig. 2. Input images used in experiments. a) Baboon. b) Monalisa c) Peppers d) Watch

The error diffusion approaches are taken from the literature. The listing 3 indicates their dimensions and weights. Table II lists the approaches names and the masks dimensions.

Error diffusion method	Mask Dimensions (width x height)
Floyd and Steinberg	3 x 2
Stevenson and Arce	7 x 4
Burkes	5 x 2
Sierra	5 x 3
Stucki	5 x 3
Jarvis, Judice and Ninke	5 x 3

TABLE II
ERROR DIFFUSION METHODS USED IN EXPERIMENTS

As we have 4 input images, 6 error diffusion methos, 2 sweep modes and it generates 2 output images (grayscale and colored), we have 96 images of output. The output images are stored in the **output** folder.

IV. DISCUSSION

This section is organized in four parts. The first part analyses the output images when compared to the original input. The second part takes into consideration the differences of sweeping modes. The third one does the comparisons between the error propagation methods. To sum up, the pros and cons of the implementation.

A. Effectiveness of the technique

In order to check if the premise of the implementation was achieved, the comparison between input and output images is needed. The figures **??** places input and output images together, for all the input images used in the experiments.

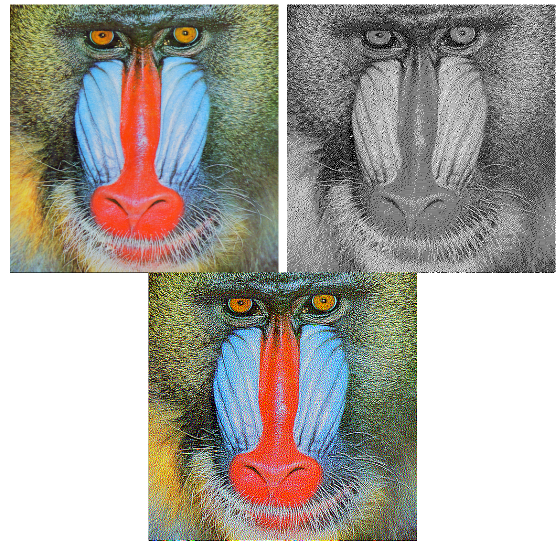

Fig. 3. Baboon images manipulated in experiments. a) Input. b) grayscale c) Peppers

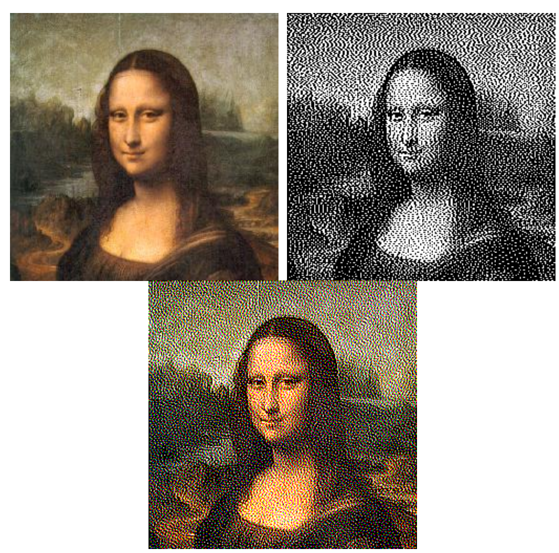

Fig. 4. Monalisa images manipulated in experiments. a) Baboon. b) Monalisa c) Peppers d) Watch

Fig. 5. Peppers images manipulated in experiments. a) Baboon. b) Monalisa c) Peppers d) Watch

Fig. 6. Watch images manipulated in experiments. a) Baboon. b) Monalisa c) Peppers d) Watch

B. Sweeping mode differences

C. Error Diffusion Methods

D. Pros and Cons

V. Conclusion