Project 1 - Halftoning

Thales 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 don't 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 $src/main.py$

ra148051@students.ic.unicamp.br

```
# 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

```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

III. Experiments

IV. Discussion

V. Conclusion