A simple amplitude modulated halftoning technique

Shankhya Debnath

August 31, 2024

1 Introduction

Halftoning is a printing technique that simulates continuous-tone images through the use of dots of varying sizes, shapes, and spacing. This technique is widely used in printing and digital imaging to reproduce photographs and other images that contain a wide range of colors or grayscales. The basic idea is to represent different shades of gray by varying the density and arrangement of black dots on white paper, thus tricking the human eye into perceiving a smooth gradient.

1.1 Why Halftoning is Necessary

The necessity of halftoning arises from the limitations of printing and display devices, which often cannot produce the continuous tone required for realistic images. Most printers, for example, can only apply ink in a binary fashion—either a dot is printed or it is not. Halftoning allows these devices to create the illusion of different shades of gray or color by adjusting the size and distribution of the dots. This process is essential for rendering high-quality images on devices that lack the ability to produce true continuous tones.

2 Mathematical Background of Halftoning

The halftoning process used in the provided code is based on a mathematical approach that involves sampling the original image and creating patterns (usually dots) that represent different intensities of color. The most common halftoning technique is the *amplitude modulation* (AM) method, where the dot size varies according to the local intensity of the image.

2.1 Dot Size Calculation

Given an input image $I(x, y)$ where x and y represent the pixel coordinates, the halftoning process involves sampling the image in small regions (sample boxes). For each sample box, the average intensity μ is calculated as:

$$\mu = \frac{1}{N} \sum_{(x,y) \in S} I(x, y)$$

where S represents the sample box containing N pixels.

The dot size d for each sample box is then determined by a non-linear function of the average intensity, typically a square root function to simulate the human eye's perception of lightness:

$$d = D_{\max}\sqrt{\frac{\mu}{255}}$$

where $D_{\max}$ is the maximum possible dot diameter corresponding to the sample size and scaling factor.

2.2 Rotation and Gray Component Replacement (GCR)

In color halftoning, each color channel (Cyan, Magenta, Yellow, and Black) is processed separately. The channels are often rotated by different angles to prevent the moiré effect, which is an undesirable artifact caused by overlapping dot patterns. The provided code allows for the adjustment of these rotation angles.

The Gray Component Replacement (GCR) technique is used to remove a portion of the gray component from the Cyan, Magenta, and Yellow channels and replace it with black ink (K channel). This technique is beneficial in reducing ink usage and improving print quality.

3 Explanation of the Code

The Python code provided implements the halftoning process for both grayscale and color images using the principles discussed above. The code allows the user to control various aspects of the halftoning process through a set of parameters.

3.1 Code Structure

The code is structured into a class called `HalftoneGenerator`, which encapsulates all the methods necessary to generate a halftone image. The primary methods and their functions are as follows:

- `__init__`: Initializes the class with the path to the input image.
- `generate_halftone`: The main method to generate the halftone image. It accepts parameters to control the halftoning process, such as sample size, scaling factor, rotation angles, and output format.
- `apply_gcr`: Applies Gray Component Replacement (GCR) to the CMYK channels.
- `create_halftone`: Generates halftone images for each color channel by calculating the dot sizes and applying rotations.
- `export_channel_images`: Saves the individual color channels as separate images, if desired.

3.2 Controllable Parameters

The code provides a variety of parameters that allow users to customize the halftoning process:

- **sample_size**: Determines the size of the sample box in pixels. Smaller sample sizes result in higher resolution but may require more processing time.
- **scaling_factor**: Controls the maximum dot size by scaling the sample size. A higher scaling factor produces larger dots.
- **gray_component_ratio**: Defines the percentage of gray to remove from the CMY channels and replace with black (K channel).
- **filename_suffix**: A suffix added to the output filename to distinguish it from the original image.
- **rotation_angles**: Specifies the angles by which to rotate each color channel to avoid the moiré effect.
- **mode**: Can be set to "color" or "grayscale" to generate a color or grayscale halftone image.
- **enable_antialiasing**: A boolean option to apply antialiasing, which smooths the edges of the dots.
- **output_file_format**: The format of the output image file, which can be "jpeg", "png", "tiff", or "default" (same as input).
- **save_individual_channels**: A boolean option to save the separate color channels (C, M, Y, K) as individual images.
- **channels_file_format**: Specifies the format of the saved channel images, similar to `output_file_format`.
- **channels_mode**: Can be set to "color" or "grayscale" for the individual channel images.

3.3 Process Overview

When the `generate_halftone` method is called, the following steps occur:

1. The input image is loaded, and the desired parameters are validated.
2. If the image is in color mode, the Gray Component Replacement (GCR) is applied to separate the CMYK channels.
3. Each channel is rotated by the specified angle, and the halftone pattern is generated by calculating dot sizes based on the sample box intensity.
4. The halftone images for each channel are optionally saved as individual files, and then merged to create the final halftone image.
5. The final image is saved in the specified output format.

4 Conclusion

Here is an example of how to use the code:

```
import halftone
h = halftone.Halftone('/path/to/image.jpg')
h.make(
    style="color",          # Can also be "grayscale"
    angles=[0, 15, 30, 45], # Use appropriate angles for each channel
    output_format="png",    # Format for the final combined image
    percentage=20,          # Percentage of gray component removal
    sample=2,               # Sample box size
    scale=5,                # Scale for dot size
    save_channels=True,     # Enable saving of channels separately
    save_channels_format="png",  # Format for the separate channel images
    save_channels_style="color"  # Style for the separate channel images
)
```