Vincenzo Macri

Digital Imaging Processing

Homework 2, Question 2:

Implement both the Floyd-Steinberg and Jarvis-Judice-Ninke dithering algorithms on the image in either Python or MATLAB, then compare the results obtained from each method.

First I will load and display the image.

```
clear; close all; clc; % clear all

originalImage = imread('cameraman.tif');
img_double = double(originalImage);

figure;
imshow(originalImage);
title('Original Image');
```

Now that the image is loaded in, I will implement both the Floyd-Steinberg and Jarvis-Judice-Ninke dithering algorithms.

1: Floyd-Steinberg Dithering

This algorithm quantizes each pixel to either 0 or 255, calculates the quantization error, and then distributes this error to neighboring pixels using the Floyd-Steinberg kernel.

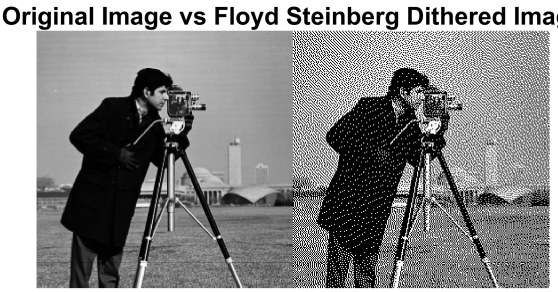
[0 0 0]

[0 * 7/16]

[3/16 5/16 1/16]

This kernel distributes the error with weights `7/16`, `3/16`, `5/16`, and `1/16` to the right and lower neighboring pixels.

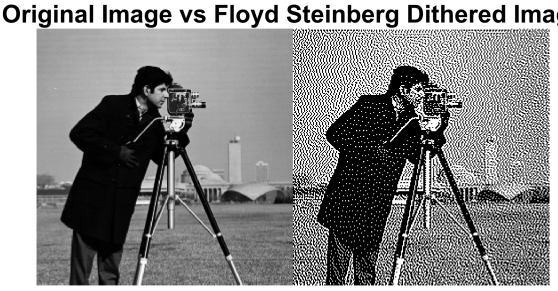
```
dither_fs = floyd_steinberg(img_double);
figure;
imshowpair(originalImage, dither_fs, 'montage');
title('Original Image vs Floyd Steinberg Dithered Image');
```



2: Jarvis-Judice-Ninke Dithering

This algorithm is similar to the Floyd-Steinberg algorithm however it uses a larger 5x3 kernel to distribute the error after quantization, resulting in smoother gradients. (See code for exact kernel values.)

```
dither_jjn = jarvis_judice_ninke(img_double);
figure;
imshowpair(originalImage, dither_jjn, 'montage');
title('Original Image vs Floyd Steinberg Dithered Image');
```



I will split my comparison into a computational perspective and a visual perspective.

From a computational perspective, Floyd-Steinberg dithering is simpler and faster due to its smaller error diffusion kernel. Jarvis-Judice-Ninke dithering, on the other hand, is more computationally intensive because it spreads the error over a larger area.

From a visual perspective, Floyd-Steinberg seems to produce more worm-like artifacts in uniform regions like the mans cloak and more dots in the mans hair and camera tripod. On the other hand, Jarvis-Judice-Ninke

results in a more evenly distributed dot pattern on the mans cloak and less dots in uniform regions like the border of the mans cloak and hair.

In short, the quality of the Jarvis-Judice-Ninke method surpasses the Floyd-Steinberg method with the tradeoff of computational speed.

```
function dithered = floyd_steinberg(img)
    [rows, cols] = size(img);
    dithered = img;
    for y = 1:rows
        for x = 1:cols
            old_pixel = dithered(y,x);
            % quantize
            new_pixel = 255 * (old_pixel > 128);
            dithered(y,x) = new_pixel;
            error = old_pixel - new_pixel;

            % distribute error
            if x+1 <= cols
                dithered(y, x+1) = dithered(y, x+1) + error * 7/16;
            end
            if y+1 <= rows
                if x-1 >=1
                    dithered(y+1, x-1) = dithered(y+1, x-1) + error * 3/16;
                end
                dithered(y+1, x) = dithered(y+1, x) + error * 5/16;
                if x+1 <= cols
                    dithered(y+1, x+1) = dithered(y+1, x+1) + error * 1/16;
                end
            end
        end
    end
    % clip vals
    dithered = (dithered > 128);
end

function dithered = jarvis_judice_ninke(img)
    [rows, cols] = size(img);
    dithered = img;
    weights = [
        0, 0, 0, 7, 5;
        3, 5, 7, 5, 3;
        1, 3, 5, 3, 1
    ];
    factor = 48;

    for y = 1:rows
        for x = 1:cols
            old_pixel = dithered(y,x);
```

```
            % quantize
            new_pixel = 255 * (old_pixel > 128);
            dithered(y,x) = new_pixel;
            error = old_pixel - new_pixel;

            % distribute error according to weights
            for dy = 0:2
                for dx = -2:2
                    new_y = y + dy;
                    new_x = x + dx;
                    if new_y > rows || new_x < 1 || new_x > cols
                        continue;
                    end
                    w = weights(dy+1, dx+3);
                    dithered(new_y, new_x) = dithered(new_y, new_x) + error * w /
factor;
                end
            end
        end
    end
    % clip vals
    dithered = (dithered > 128);
end
```