Trabalho 2 de Introdução ao Processamento de Imagem Digital

Nome: Paulo Junio Reis Rodrigues - RA: 265674

13/11/2020

1 Introdução

O objetivo deste trabalho, será aplicar o método de meios-tons (halftoning), muito utilizado em impressões de imagens em jornais e revistas. Esta técnica, utiliza um método para pontilhar a imagem em pontos pretos e brancos para reduzir o número de níveis de cinza da imagem. Em figuras coloridas, o método é aplicado em cada camada de cor, como por exemplo em uma imagem RGB.

No caso deste trabalho, será feito o pontilhamento com difusão de erro, que consiste em distribuir a diferença entre o valor exato de cada pixel e seu valor aproximado a um conjunto de pixels adjacentes, algumas propostas de distribuições de erro podem ser vistas na figura 1. Ao final, depois da utilização da técnica, é gerada uma nova imagem com apenas dois valores de pixels (0 e 255).

Portanto, o intuito deste trabalho é, a partir de várias distribuições de erro gerar várias imagens coloridas e monocromáticas, para comparar todos os filtros utilizados, e também comparar a maneira que eles foram aplicados (unidirecionalmente ou alternadamente).

Junto com este relatório, está sendo enviado o arquivo trabalho_2_265674.zip que possui dentro dele todos os arquivos que irão ser citados neste relatório.

	$f(x,y)$	7/16
3/16	5/16	1/16

(a) Floyd e Steinberg

			$f(x,y)$		32/200	
12/200		26/200		30/200		16/200
	12/200		26/200		12/200	
5/200		12/200		12/200		5/200

(b) Stevenson e Arce

		$f(x,y)$	8/32	4/32
2/32	4/32	8/32	4/32	2/32

(c) Burkes

		$f(x,y)$	5/32	3/32
2/32	4/32	5/32	4/32	2/32
	2/32	3/32	2/32	

(d) Sierra

		$f(x,y)$	8/42	4/42
2/42	4/42	8/42	4/42	2/42
1/42	2/42	4/42	2/42	1/42

(e) Stucki

		$f(x,y)$	7/48	5/48
3/48	5/48	7/48	5/48	3/48
1/48	3/48	5/48	3/48	1/48

(f) Jarvis, Judice e Ninke

Figura 1: Distribuições de erro, onde $f(x,y)$ é o pixel no qual está sendo aplicado a difusão.

2 Programa

Todos os programas foram implementados usando Python 3.8.5, com as bibliotecas Numpy 1.19.2 e OpenCV 4.4.0.42.

2.1 Como executar

O programa pode ser executado através do script *trabalho_2.py*. Quando executado, obrigatoriamente deve ser enviado a uma imagem colorida, porém caso queira, poderá ser enviado como argumento, dois valores:

```
--alternado: Ativa a opção de realizar o percurso do meio-tons de forma alternada.

--monocromatico: Ativa a opção de realizar a leitura da imagem em escala de cinza.
```

Logo abaixo é mostrado um exemplo de execução do script:

```
python trabalho_2.py imagens/baboon.png --alternado --monocromatico
```

2.2 Entrada

O programa possui 3 entradas, a primeira é a imagem que será utilizada para aplicar o método, a segunda é a opção de realizar o percurso da imagem de forma alternada e por último tem-se a opção de ler a imagem recebida em escala de cinza.

2.3 Saída

As saídas do programa são 6 imagens coloridas no formato *PNG*, uma para cada distribuição de erro aplicada. Porém caso a opção de leitura monocromática seja ativada, ao final da execução do script, será reproduzida 12 imagens no formato *PNG*, sendo 6 delas coloridas, e outras 6 imagens monocromáticas.

3 Parâmetros utilizados

Todas as imagens utilizadas para execução do programa estão presentes no diretório imagens/, e todas estão disponíveis em *https://www.ic.unicamp.br/ helio/imagens_coloridas/*.

As saídas do script executado, terão o nome do trabalho(trabalho_2) e logo depois o nome da distribuição utilizada, exemplo: trabalho_2_FloydSteinberg.png, caso a resposta seja em escala de cinza, ao final do nome será colocada a palavra Cinza, exemplo: trabalho_2_FloydSteinbergCinza.png, além disso, caso a opção de percurso alternado estiver ativado, no final do nome do arquivo será colocado a palavra "alternado", exemplo: trabalho_2_FloydSteinbergCinza_alternado.png e todas estas saídas estarão no diretório outputs/.

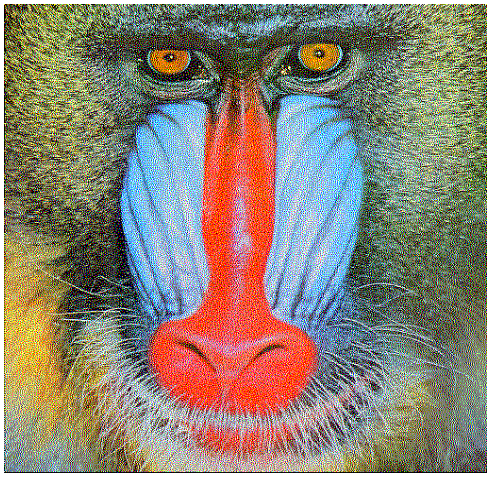
4 Solução do exercício

A solução do trabalho utiliza a função *imread* do *OpenCV* para a leitura das imagens a partir do diretório informado. As imagens serão lidas via camada de cores (BGR), porém caso a opção monocromática esteja ativada, o programa irá ler as imagens via escala de cinza. Todo o código foi baseado no pseudocódigo apresentado no material do professor.

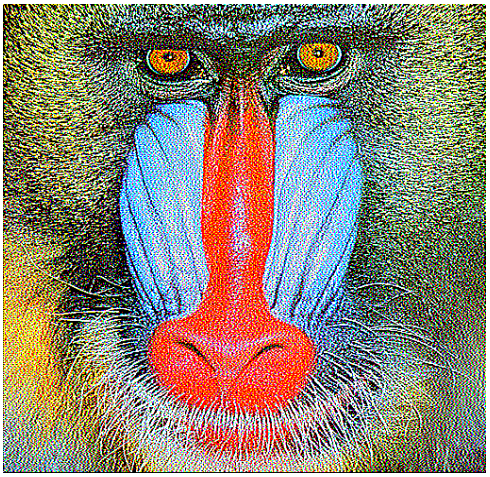
Logo após a imagem ser carregada, o algoritmo aplica um tratamento de bordas antes da aplicação do meio-tons, para este trabalho, será utilizado o tratamento de bordas por uma constantes, que no caso será o valor 0, onde consiste em aplicar em todos os pixels das bordas horizontais e verticais o valor 0, para criar uma camada maior das bordas, para que no processo de meio-tons não ocorra nenhum erro. Logo depois deste tratamento o programa transforma os valores da imagem original em decimais (*Float64*), utilizando a biblioteca *Numpy*, para que no momento que o algoritmo distribuir o erro para os pixels subsequentes, não ocorra nenhum problema nos cálculos.

Depois destes processos, o algoritmo começa a verificar cada pixel de forma unidirecional ou alternado dependo da forma escolhida nos argumentos, e para cada valor menor que 128 encontrado, o valor atribuído ao pixel da imagem resultado é igual a 0, e para valores maiores ou iguais a 128 o

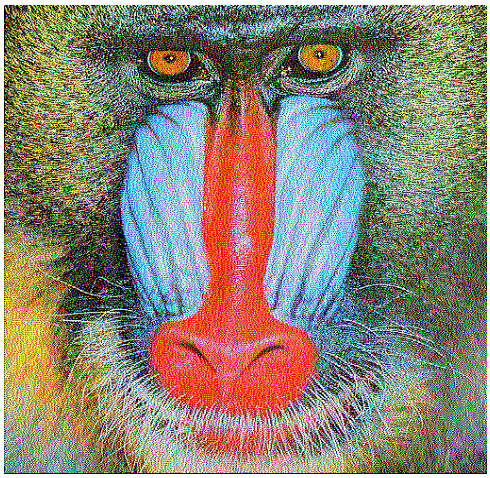
valor atribuído ao pixel é igual a 255, isso é feito em cada camada de cor da imagem. Em seguida, é feito o cálculo do erro (imagem original - imagem resultante), para que, junto com a aplicação dos filtros na imagem, seja feita a sua multiplicação, fazendo com que o erro seja propagado para os pixels subsequentes.



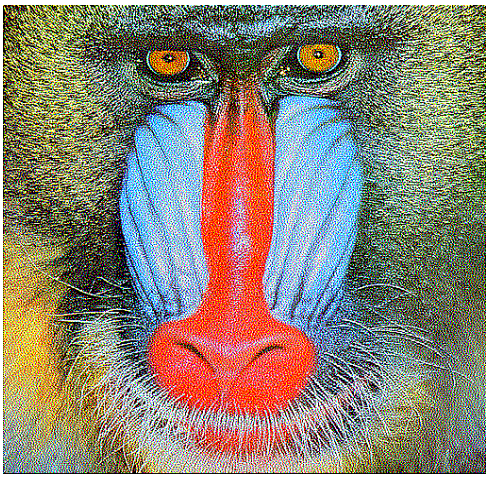
(a) Floyd e Steinberg



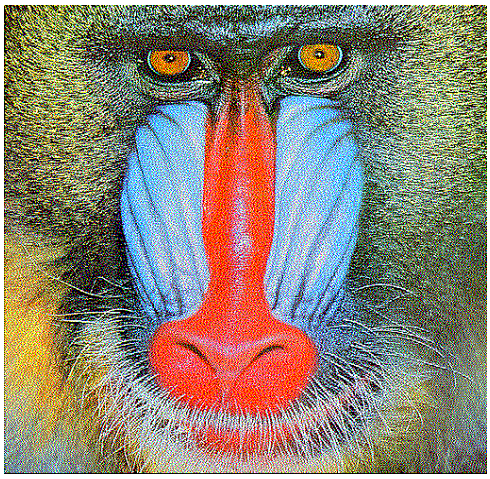
(b) Stevenson e Arce



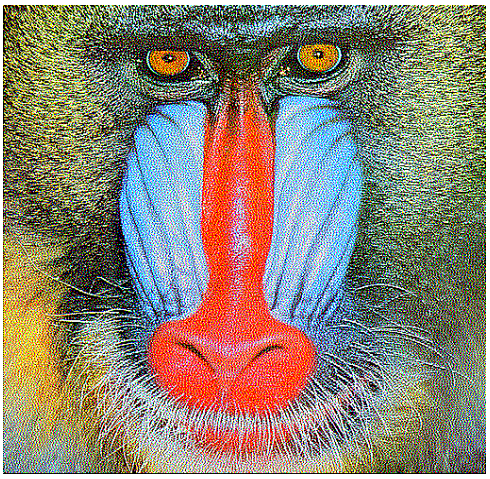
(c) Burkes



(d) Sierra



(e) Stucki



(f) Jarvis, Judice e Ninke

Figura 2: Aplicações do meio-tons com varias difusões de erro na imagem do baboon.

Ao final da execução, uma imagem resultante é gerada, com somente valores de 0 e 255. Na fi-

Figura 3: Imagem original do baboon.

gura 2, pode ser observado alguns exemplos de aplicação deste método, para diferentes distribuições de erro.

Quando comparada a imagem original, as imagens pontilhadas geradas obtiveram um resultado bem parecidos com a imagem original, entretanto, alguns ruídos foram gerados. Em destaque, a distribuição de erro de *Stevenson e Arce* obteve mais ruído que as demais.

O efeito meio-tons é perceptível nas imagens geradas, pois a figura gerada possui somente dois valores (0 e 255) em cada camada de cor, fazendo com que a imagem esteja sendo representado por uma gama de cores menor, deixando a imagem com pontos mais brancos por exemplo.

4.1 Comparação das distribuições de erro

Para uma melhor comparação das distribuições, foi feita a execução do método utilizando a função monocromática. Todos os resultados foram feitos na forma alternada e estão representados na figura 4. Através destas imagens a percepção do efeito meio-tons é bastante percebida, pois, como agora só possui dois valores (preto e branco), é mais fácil visualizar a diferença entre os filtros.

A primeira percepção é clara, o filtro do **Stevenson e Arce** possui muito mais ruído do que os outros, isso faz com que o efeito meio-tons não seja muito efetivo na imagem, deixando o resultado com tons de cinza diferentes da imagem original.

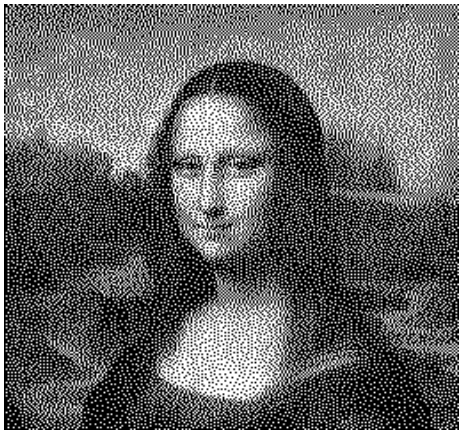
Agora outro ponto percebido foi nos olhos da Monalisa, nos filtros de **Sierra** e **Stucki** é mais nítido perceber que a parte branca dos olhos é melhor representada, já nos outros filtros, como o **Burkes** por exemplo, não possuem uma boa representação da parte branca dos olhos.

Outro ponto bem diferente na imagem é o pescoço da Monalisa, no filtro de **Sierra** e o filtro do **Jarvis, Judice e Ninke** é o possível perceber que eles possuem uma melhor harmonização dos tons de cinza no local, representando um bom sombreamento. Entretanto, os outros filtros não conseguiram o mesmo efeito, deixando o sombreamento mal representado neste local.

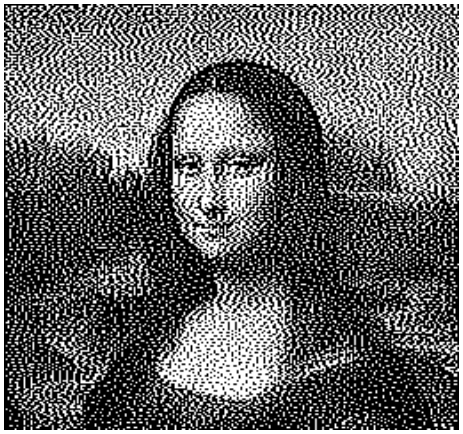
Contudo, no caso desta imagem, o melhor resultado entre os 6 filtros, foi o filtro de **Jarvis, Judice e Ninke**, ele representou fielmente a imagem original, deixando principalmente as sombras mais bem definidas.

4.2 Comparação entre as varreduras

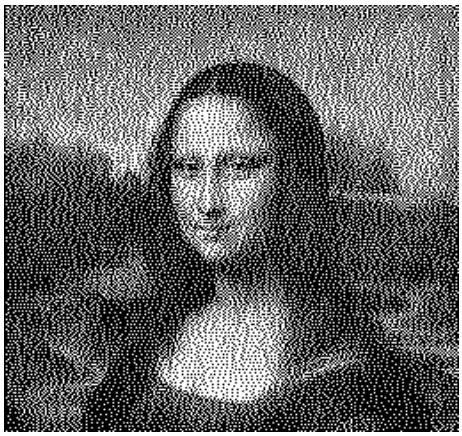
Como já foi dito na explicação do algoritmo, o programa possui duas opções de percurso, o unidirecional e a outra alternada. Não há muita diferença a olho nu entre as duas varreduras, porém, quando é aplicado um zoom na imagem do **baboon** monocromática, é possível perceber que na imagem unidirecional a textura do rosto do **baboon** possui algumas linhas pretas em direção à diagonal principal, e já na varredura alternada essas linhas são mais retas. Neste caso o filtro utilizado foi do **Jarvis, Judice e Ninke**, e o resultado pode ser observado na figura 5.



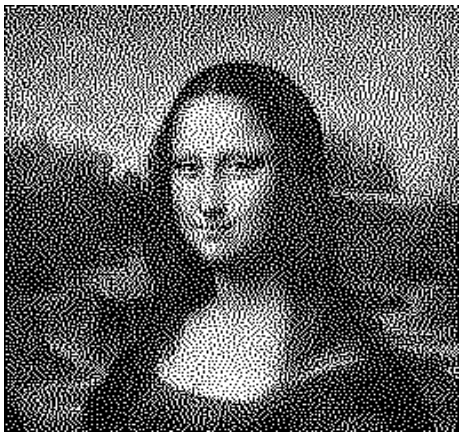
(a) Floyd e Steinberg



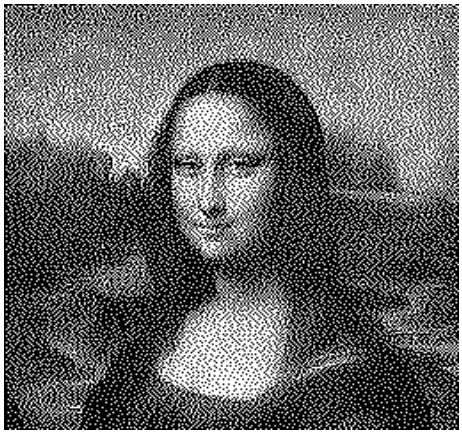
(b) Stevenson e Arce



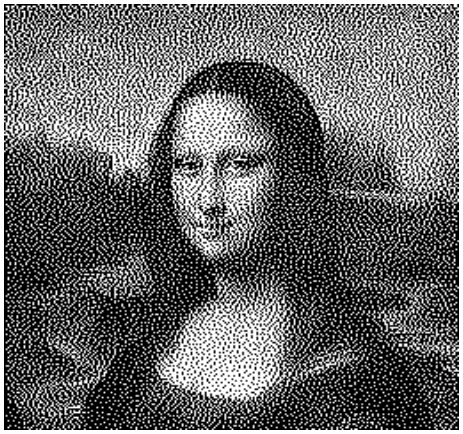
(c) Burkes



(d) Sierra



(e) Stucki

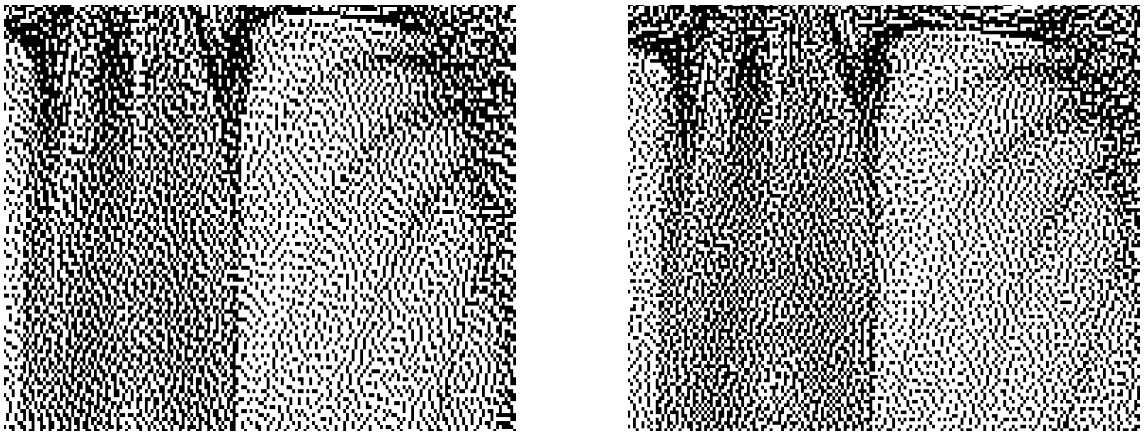


(f) Jarvis, Judice e Ninke

Figura 4: Aplicações do pontilhamento com varias difusões de erro na imagem da Monalisa em tons de cinza.

5 Limitações

Para este trabalho foi usado como exemplo o pseudocódigo do professor, que neste caso, possui uma baixo desempenho, dependendo bastante da imagem que será utilizada na execução do script. Isso se dá pelo fato de que o algoritmo não pode ser vetorizado, pois as operações (distribuições de erro) realizadas necessitam ser mantidas durante a execução, deixando inviável a vetorização deste método. Contudo, caso este algoritmo seja utilizado para imagens grandes, o programa pode demorar para entregar os resultados.



(a) Unidirecional (b) Alternado

Figura 5: Imagem do baboon monocromática com a varredura unidirecional e alternada.