Projeto 1 - MC920A - Técnica de meios-tons

Marcelo Biagi Martins - 183303

1 Introdução

A técnica de aplicação de filtros em imagens é uma prática antiga, que surge quase que concomitantemente com o conceito de processamento de imagens [1]. Com essa técnica, diversos objetivos podem ser alcançados. Desde técnicos, como diminuição da memória utilizada e aumento da velocidade de processamento; até práticos, como alteração de brilho e contraste para facilitar a visualização de uma placa de carro acima da velocidade permitida em uma rodovia, algo que é utilizado nos radares.

Assim, neste contexto estão inclusos as técnicas de meios-tons [2] que serão exploradas nesse projeto. Essas técnicas tem como função alterar o número de tons de cinza de uma imagem [3], tornando menor a quantidade de cores necessárias para a visualização da mesma. Sua função é essencial em meios de comunicação como jornais, que apenas utilizam as cores preto e branco, em geral.

Esse trabalho implementa seis técnicas de pontilhado com difusão de erro: Floyd, Stevenson, Burkes, Sierra, Stucki e Jarvis e duas formas de varredura de imagem: straight e snake. Cada combinação entre essas técnicas possui valores e tamanhos distintos que implicam diretamente no resultado final.

2 Especificações do programa

O script em python se chama proj1.py e, para executá-lo, basta rodar o comando "python3 proj1.py *diretórioDasImagens option*"com *option=5* que todas as combinações serão executadas diretamente para todas as imagens do diretório especificado no argumento diretórioDaImagem, na ordem: snakeColorida + filtros, straightColorida + filtros, snakeCinza + filtros, straightCinza + filtros. Assim, todas as combinações possíveis são geradas de forma automática. Caso, seja de interesse do usuário executar outra ordem, basta alterar o último argumento do comando de execução, da forma descrita na tabela 1. Importante notar que todas as opções geram todos os filtros para a(s) imagens.

Tabela 1: **Parâmetro numérico de "python3 proj1.py option"**

option	Executa
1	Snake Colorida
2	Straight Colorida
3	Snake Cinza
4	Straight Cinza
5	Todos
Outro	Inválido

As saídas são as imagens com as técnicas já aplicadas e com nome na forma: "*nome da imagem*"+ "*color*"/"*gray*"+ "*Snake*"/"*Straight*"+ "*nome do filtro*"+ "*.png*". Assim, o script é limitado à leitura

no formato png, além de todas as suas saídas manterem esse padrão. As imagens geradas, por sua vez, são salvas no mesmo diretório que o script estiver.

Quanto às bibliotecas, foram utilizadas neste projeto: *opencv2*, para leitura e escrita de imagens; *matplotlib*, para plot das imagens na fase de desenvolvimento, algo que não é mais necessário na fase final; *numpy*, para manipulação das imagens; e *os* e *glob*, para executar uma função que armazena todas as imagens e seus respectivos nomes de um diretório para automatizar o processo e executar tudo de forma direta.

3 Funcionamento do programa

Importante ressaltar que o script emite um *RunTimeWarning* de overflow na primeira vez que encontra a variável *err* das funções que percorrem a imagem. Esse erro vêm do fato de que as imagens abertas com o OpenCv2 têm seus pixels no formato *uint-8*, ou seja, são inteiros de 8 bits. Assim, ao somarmos 255 com 2, temos 1 como resultado e não 257. Portanto, este warning emitido apenas nos alerta desse fato e não altera o funcionamento do script, que depende dessa propriedade dos *uint-8* para funcionar.

O script possui uma função main que controla sua execução. Primeiramente, todos os filtros foram previamente criados como numpy arrays e transformados em matrizes com a função *numpy.reshape()* com as devidas proporções. Posteriormente, o parâmetro *option* é lido e a ação correspondente é tomada. A partir deste ponto, o fluxo de execução para a opção 1 - colorido snake, por exemplo, é:

- Os vetores com as imagens e seus nomes são gerados
- A devida função *run(Straight/Snakes)(Color/Gray)* é chamada para cada imagem, utilizando todos os filtros
- Neste caso colorido-snake, segue-se para a função *runSnakesColor*
- Essa função possui o laço que vai testar todas os filtros para todas as imagens e, para cada par, chama a *testSnakesColor*
- Na *testSnakesColor*, para cada combinação imagem-filtro que vier, o procedimento será: split da matriz em R, G, B; criação de uma borda de tamanho 5 para cada componente e chamada da função *textSnakeColor* para cada um
- Assim, cada componente se tornará binário através da execução da *goThroughSnake*, que implementa uma técnica de meios-tons com difusão de erro genérica, que altera seus fatores de multiplicação de acordo com o filtro, distribuindo o erro entre os vizinhos.
- Ao retornar, as três componentes binárias são agora sobrepostas com o uso da função *cv2.merge()* e sua borda é cortada de forma vetorizada
- Por fim, a função *runSnakesColor* salva a imagem no mesmo diretório do script com o nome adequado, que neste caso, será *'NameColorSnakeFilter.png'*, em que *Name* é o nome da imagem original e *Filter* é o nome do filtro
- Após a execução de todas as combinações possíveis conforme a opção escolhida, a execução termina

Esse fluxo é o mesmo para todas as opções, mudando apenas as funções chamadas. Apesar de ser possível juntar funções como: *testStraightColor* e *testSnakesColor* em uma só, apenas adicionando uma flag como parâmetro de diferenciação, a separação foi escolhida para facilitar o entendimento, visto que a velocidade de execução praticamente não é afetada com essa mudança.

4 Resultados

É importante notar que as técnicas de meios-tons utilizadas neste projeto implementam a técnica de difusão de erro, visando diminuir a diferença entre os vizinhos de forma que a imagem fique o mais parecido possível com a original de 256 tons de cinza, mesmo sendo binária. Como são gerados 12 imagens para cada imagem base, torna-se inviável demonstrar todos os resultados neste relatório. No entanto, pode-se fazer algumas comparações pertinentes. A primeira sendo a diferença entre o caminho Snake e o Straight. Na figura 1 pode-se analisar a diferença entre Snake e Straight, respectivamente, para a imagem do babuíno fornecida, todas para o filtro Floyd.

Enquanto que na escala de cinza a diferença é delicada, esta torna-se mais visível na escala colorida. É marcante como o nariz do babuíno na imagem gerada com a forma Straight possui mais pixels incoerentes com seus vizinhos. Isso se dá devido ao fato de que na forma Snake, além do erro ser distribuído entre os vizinhos, ele também muda de orientação, pois ao mudarmos o sentido com que estamos percorrendo a linha da matriz, também giramos o filtro 180º na horizontal, tornando os erros ainda mais espalhados e discretos. Como na forma Straight isso não acontece, partes com cor constante como o nariz acabam acumulando o erro de cada componente R, G, B e, ao dar merge das três, cores discrepantes aparecem. Na escala de cinza não temos o mesmo problema de forma tão marcante, pois apesar da fonte de erro ainda estar presente, apenas temos uma componente e assim, o erro total se torna menor.

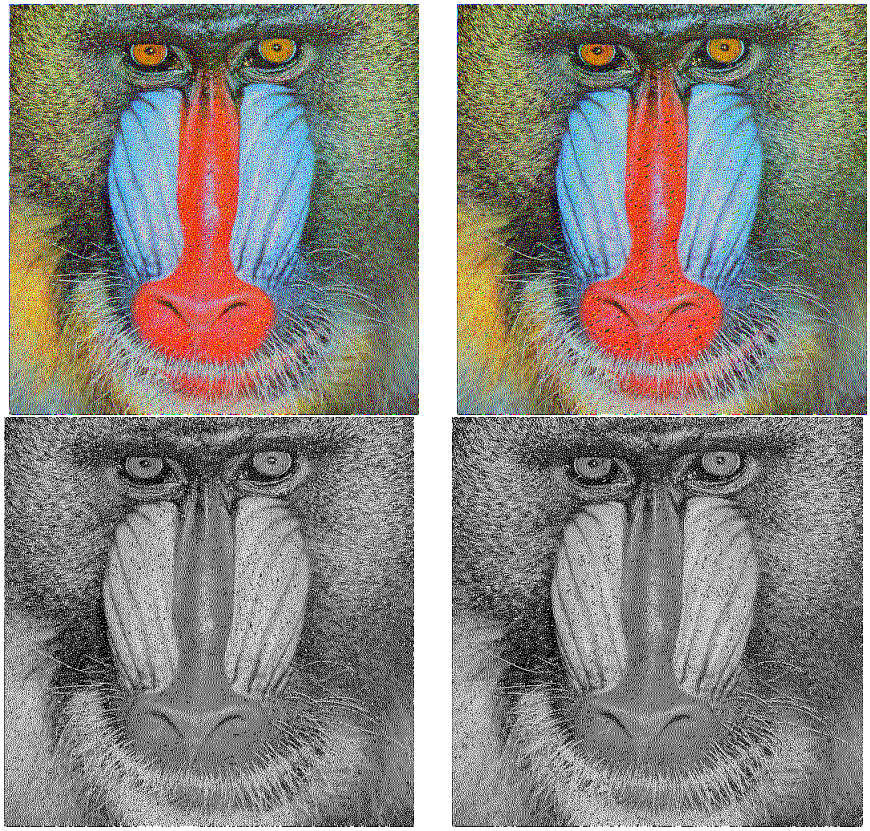

Figura 1: Comparação entre Snake e Straight para imagem colorida e cinza de um babuíno

O mesmo teste pode ser feito com outra imagem, como na figura 3. No entanto, como agora a imagem passa de 512x512 para 256x256, o erro se torna menos visível, mas ainda presente e notável se observarmos o rosto ou o fundo da figura, que apresentam cores constantes. Em resumo, as mesmas conclusões do babuíno podem ser tomadas aqui.

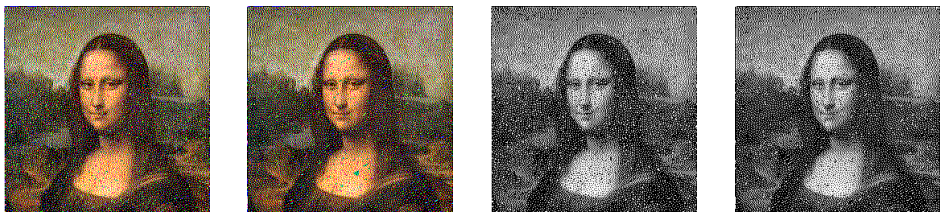

Figura 2: Comparação entre Snake e Straight para imagem colorida e cinza da Monalisa

Outra comparação pertinente é a diferença entre filtros. Como os erros são mais visíveis em imagens coloridas, essas serão o foco deste teste. Ainda, para manter um padrão, todas as imagens deste teste serão percorridas com o modo Snake. Utilizando uma mesma imagem, podemos notar a diferença entre o menor dos filtros(Floyd), um médio(Sierra) e o maior(Stevenson).

Figura 3: Análise da influência do tamanho do filtro no resultado final. Floyd, Sierra e Stevenson

É visível que o filtro de Stevenson, por ser o maior de todos, possui uma distruibuição mais linear dos erros, tornando a sua imagem com mais detalhes do que as outras, além de possuir cores mais coerentes do que outros filtros, como na parte superior da figura onde as correntes do relógio estão presas. Assim, um filtro grande tem a capacidade de gerar imagens mais fidedignas as originais. No entanto, ao mesmo tempo ele acaba gerando uma imagem mais "quadriculada", algo que é visível em imagens menores, como a da Monalisa. Assim, sua utilização só é eficiente em certas condições. Além disso, este filtro 4x7 realiza 28 operações de multiplicação para cada pixel da imagem, enquanto que o de Floyd realiza 2x3 = 6. Assim, é previsível que quanto maior o filtro, mais tempo levará a sua execução. Este é um grande tradeoff dessa técnica e algo que precisa ser analisado de acordo com o contexto. Podemos concluir que a vantagem dos filtros maiores está na geração de imagens mais nítidas, enquanto que a vantagem dos filtros pequenos é a sua velocidade maior de execução.

Outra comparação relevante possível que pode ser feita é a análise da diferença dos quocientes dos filtros no resultado final. Dos dados utilizados, percebe-se que os filtros de *stucki, sierra* e *jarvis* possuem as mesmas proporções, mas quoficientes de multiplicação diferentes. No entanto, essa variação é relativamente baixa. No geral, os coeficientes stucki são 17% maiores do que os sierra. Enquanto que a relação sierra/jarvis e stucki/jarvis variam no máximo 7%. Como os valores são dos coeficientes são muito baixos, essa porcentagem acaba mudando muito pouco o valor na prática.

Para uma melhor análise, a figura 4 mostra a diferença dos resultados da foto do babuíno com modo snake para os filtros stucki, sierra e jarvis, respectivamente. Como era esperado, essa mudança tem pouquissímos efeitos visíveis a olho nu e não é possível distinguir entre os filtros.

Por fim, é interessante comparar todos os filtros para uma imagem fixa. Para isso, a figura 5

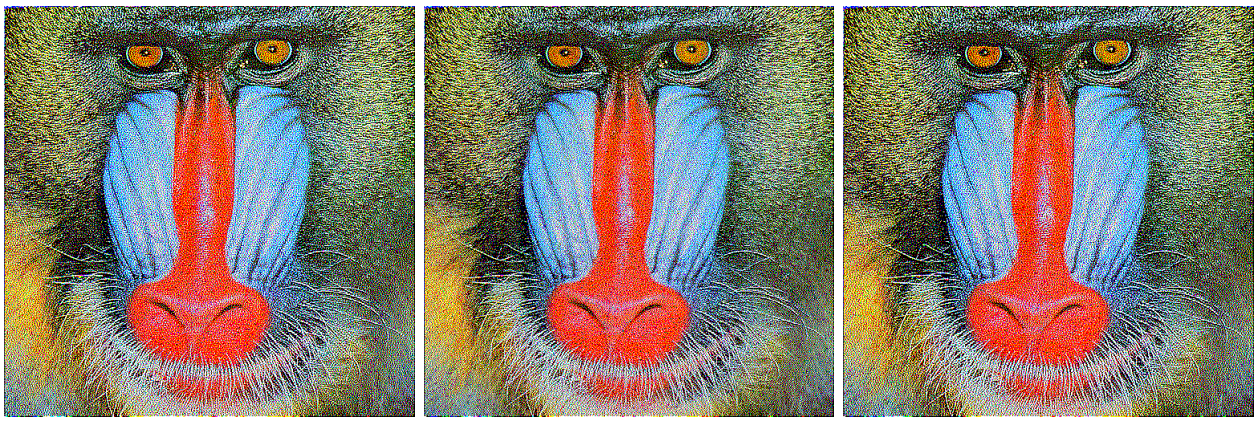

Figura 4: Análise de diferença entre coeficientes de filtros de mesma proporção

compara todos os filtros para a figura da Monalisa no modo snake colorido. Notamos que os filtros de tamanho médio(burkes, sierra, stucki e jarvis) apresentaram um resultado melhor, mas a diferença ainda assim é pouca. É notável a pixelização do resultado de stevenson e um pouco de ruído no de floyd.

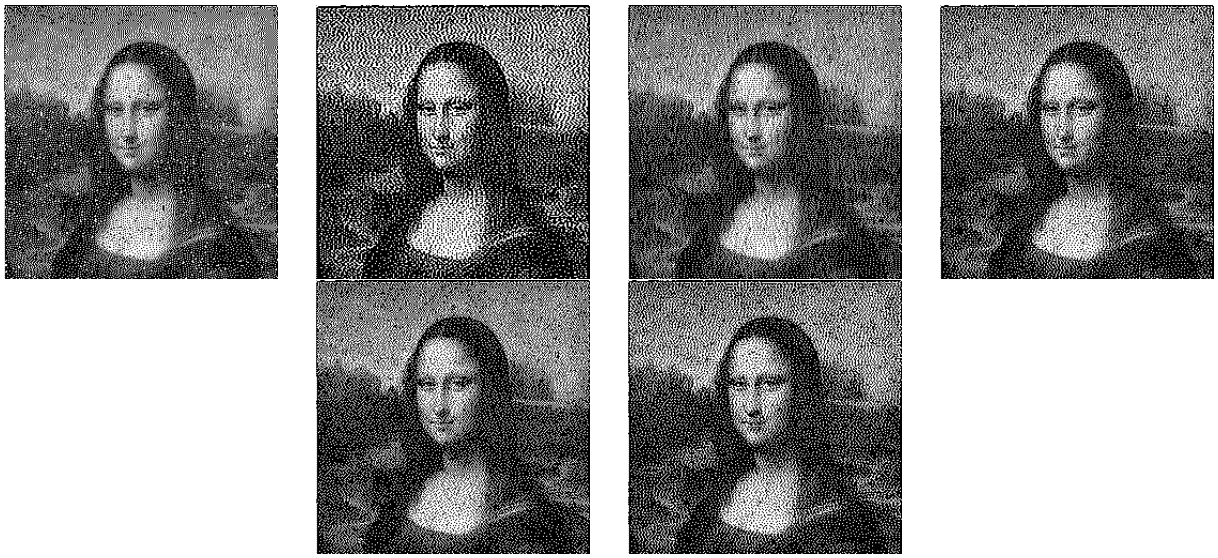

Figura 5: Monalisas: floyd, stevenson, burkes, sierra, stucki, jarvis

5 Conclusões

É marcante o fato de que as técnicas de meios-tons são muito poderosas e tem grande aplicação no mundo fora dos livros. A economia de cores proporciona grandes economias de recursos, como tintas de impressora, além do fato de que meios de comunicação com cores limitadas, como os jornais, se beneficiam muito desta técnica.

Neste projeto pode-se analisar melhor os resultados dessas técnicas e de suas variações. Foi possível perceber as diferenças entre diversas medidas na aplicação dos filtros e seus efeitos, como proporção, resolução e tempo de execução. Os resultados mais impressionante são os feitos com filtros maiores, enquanto que os menores ainda assim fornecem imagens com qualidade e em tempo de execução menor.

Quanto aos coeficientes, suas diferenças não são muito vísiveis se a mudança de valor for baixa, como é o caso neste projeto. Por fim, é marcante como a forma de visualização da imagem altera seu resultado, sendo necessário um bom visualizador para conseguir enxergar as nuances e os detalhes modificados.

Referências

[1] *O que é Image Filtering?*, Em https://www.mathworks.com/help/images/what-is-image-filtering-in-the-spatial-domain.html 1

[2] *Aula Realce*, Em http://www.ic.unicamp.br/~helio/disciplinas/MC920/aula_realce.pdf 1

[3] *About dithering*, Em http://www.ece.ubc.ca/~irenek/techpaps/introip/manual04.html 1