UFF – Universidade Federal Fluminense
TIC – Instituto de Computação
TCC – Departamento de Ciência da Computação

Disciplina: TCC 00.174 Programação de Computadores II / Turma: A-1 / 2014.2
Professor: Leandro Augusto Frata Fernandes

Trabalho - Photochopp | Divulgação: 06/09/2014 | Atualizado em: 29/09/2014 | Entrega: vide cronograma

Instruções

1) A entrega do trabalho consiste no envio do código fonte Java organizado em um projeto NetBeans® autocontido, i.e., que não dependa de ou inclua pacotes externo às APIs que compõe o Java SDK.
2) Serão considerados entregues trabalhos enviados para o e-mail de seu respectivo professor (laffernandes@ic.uff.br / laffernandes@gmail.com) até as 23:59 da data de entrega definida no cronograma. A única exceção a esta regra é o caso a conta de e-mail de seu professor estar fora do ar. Será atribuída nota ZERO a trabalhos não entregues. Trabalhos entregues além deste prazo serão considerados não entregues, mesmo que por questão de minutos.
3) Projeto NetBeans® que não compila por erros de sintaxe no código caracteriza incapacidade de produzir programas. Para esses será atribuída a nota ZERO, mesmo que parte do código possa ser validada por teste de mesa.
4) Este trabalho é individual. Trabalhos entregues com código-fonte idêntico, ou com reformatações (e.g., alteração de nome de variáveis, métodos ou classes) terão nota final igual à ZERO.
5) Leia atentamente o enunciado antes de proceder com as implementações. Texto colorido representa porções atualizadas do enunciado.

Enunciado

Você deverá criar um pacote que conterá classes responsáveis pela aplicação de operações simples em imagens oriundas de arquivos de imagens. Este pacote será utilizado por um aplicativo Java escrito por você e que, uma vez pronto, será invocado pelo usuário a partir do prompt de comando do DOS ou Linux.

Com o intuito de facilitar a carga, manipulação e salvamento dos arquivos de imagem, está autorizado o uso das classes: `javax.imageio.ImageIO`, `java.awt.image.BufferedImage` e `java.io.File`. Segue um exemplo simples de como essas classes podem ser utilizadas para: (1) abrir um arquivo de imagem; (2) trocar a cor dos três primeiros pixels da primeira linha para, respectivamente, azul, verde e vermelho; e (3) salvar a imagem em um arquivo diferente do primeiro.

```
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

public class ExemploSimples {

    public static void main(String[] args) throws IOException {
        BufferedImage img = ImageIO.read(new File("imagem_original.png"));

        final int blue = 0x000000FF;  // Assumindo image.getType() ==
        final int green = 0x0000FF00; //        BufferedImage.TYPE_3BYTE_BGR
        final int red = 0x00FF0000;

        img.setRGB(0, 0, blue);
        img.setRGB(1, 0, green);
        img.setRGB(2, 0, red);

        ImageIO.write(img, "png", new File("imagem_modificada.png"));
    }
}
```

As recomendações específicas para este trabalho são:

a) Com exceção da classe principal da aplicação, todas as outras classes deverão estar contidas no pacote `photochopp`, uma alternativa de baixo custo à ferramenta Adobe Photoshop®. A classe principal deverá ser implementada no pacote `app` e receberá o nome `Trabalho`.

b) Linhas gerais para definição das classes são apresentadas do decorrer do enunciado. Seguindo a linha geral, crie quantas classes julgar necessário. A capacidade de organização do código faz parte da avaliação.

c) O pacote deve implementar a operação de complemento de imagens coloridas (espaço de cores RGB ou BGR). O complemento é calculado utilizando a seguinte função:

$$R_C = 255 - R, \quad G_C = 255 - G, \quad B_C = 255 - B$$

onde R, G e B são, respectivamente, as intensidades dos canais vermelho, verde e azul de um dado pixel da imagem de entrada e R_C, G_C e B_C são, respectivamente, as intensidades dos canais vermelho, verde e azul do pixel na imagem resultante. Aqui é assumido que os canais da imagem colorida assumem valores inteiros no intervalo $[0, 255]$.

A operação de complemento de imagens deverá ser implementada em um método estático da classe `photochopp.Operacoes`. Ele deverá receber como único argumento a instância de `BufferedImage` que contém a imagem de entrada e retornar outra instância de `BufferedImage` que conterá a imagem de saída.

d) O pacote deve implementar a operação de conversão de imagens coloridas (espaço de cores RGB ou BGR) em imagens em tons de cinza. A conversão é feita utilizando a seguinte função:

$$I = 0.3\,R + 0.59\,G + 0.11\,B$$

onde R, G e B são, respectivamente, as intensidades dos canais vermelho, verde e azul de um dado pixel da imagem de entrada, e I é a intensidade do tom de cinza resultante da conversão.

A operação de conversão de RGB para tons de cinza deverá ser implementada em um método estático da classe `photochopp.Operacoes`. Ele deverá receber como único argumento a instância de `BufferedImage` que contém a imagem de entrada e retornar outra instância de `BufferedImage` que conterá a imagem de saída.

e) O pacote deve implementar o operador de *convolução discreta 2D*, um dos operadores fundamentais em processamento de imagens (exemplos de aplicações serão dados na sequência).

Convolução é um operador matemático (denotado por $*$) que age sobre duas funções, f e w, produzindo uma terceira função. No caso 2D o operador de convolução é dado por:

$$(f * w)(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x - u, y - v)\, w(u, v)\, du\, dv$$

Convolução discreta nada mais é do que a aplicação do operador de convolução sobre funções definidas em um domínio discreto. Quando essas funções são bidimensionais (e.g., imagens raster, que são imagens representadas por uma grade regular de pixels), o operador de convolução discreta é dado por:

$$(f * w)(x, y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(x - u, y - v)\, w(u, v)$$

Felizmente, ao trabalhar com imagens raster, os somatórios não precisam ser avaliados com índices variando no intervalo $[-\infty, +\infty]$, pois a quantidade de linhas e colunas contidas em uma imagem é finita. Em processamento de imagens a simplificação da expressão do operador de convolução costuma ser ainda maior, pois a função f costuma ser a imagem sobre a qual queremos aplicar algum filtro linear, e w uma matriz quadrada de dimensões $K \times K$ (para K ímpar)

que é muito menor que f. No contexto de processamento de imagens, w codifica um filtro linear a ser aplicado sobre f. Neste caso, a expressão da convolução discreta 2D reduz para:

$$(f * w)(x,y) = \sum_{u=-K/2}^{+K/2} \sum_{v=-K/2}^{+K/2} f(x-u, y-v)\, w(u,v)$$

Fica implícito nesta definição que $K/2$ retorna a parte inteira do resultado da divisão e que as coordenadas dos elementos na matriz w são endereçadas por $u \in \left[-\frac{K}{2}, -\frac{K}{2}+1, \ldots, 0, \ldots, +\frac{K}{2}-1, \frac{K}{2}\right]$ e $v \in \left[-\frac{K}{2}, -\frac{K}{2}+1, \ldots, 0, \ldots, +\frac{K}{2}-1, \frac{K}{2}\right]$; ao passo que os elementos (pixels) na imagem f são endereçados utilizando qualquer convenção. Por simplicidade é sugerido o uso de $x \in [0, 1, \ldots, N-1]$ e $y \in [0, 1, \ldots, M-1]$, onde N e M são, respectivamente, a largura e altura da imagem raster f.

Segue a representação gráfica da aplicação de um filtro w de tamanho 3×3 sobre o pixel de coordenadas $x = 4$ e $y = 3$ no canal R de uma imagem f de tamanho 5×8. A parte em cinza em f_r corresponde à área de atuação do filtro quando aplicado sobre o pixel tratado. O resultado da aplicação do filtro é atribuído ao pixel de coordenadas $x = 4$ e $y = 3$ da imagem $g = f * w$.

Filtro w

		u = -1	u = 0	u = 1
	-1	1.0	2.0	3.0
v	0	4.0	5.0	6.0
	1	7.0	8.0	9.0

Imagem f_r

		x = 0	1	2	3	4	5	6	7
	0	255	30	25	40	60	78	96	125
	1	128	7	14	32	96	100	48	99
y	2	0	204	9	70	78	68	128	64
	3	2	5	9	69	67	97	0	78
	4	7	78	201	0	102	77	12	48

Imagem g_r

y \ x	0	1	2	3	4	5	6	7
0								
1								
2								
3					3148			
4								

$$g_r(4,3) = (f_r * w)(4,3) = \sum_{u=-1}^{+1} \sum_{v=-1}^{+1} f_r(4-u, 3-v)\, w(u,v)$$
$$= f_r(5,4)\, w(-1,-1) + f_r(4,4)\, w(0,-1) + f_r(3,4)\, w(1,-1) + f_r(5,3)\, w(-1,0)$$
$$+ f_r(4,3)\, w(0,0) + f_r(3,3)\, w(1,0) + f_r(3,2)\, w(-1,1) + f_r(4,2)\, w(0,1) + f_r(3,2)\, w(1,1)$$
$$= 3148$$

A operação de convolução discreta 2D deverá ser implementada em um método estático da classe `photochopp.Operacoes`. Ele deverá receber como argumento a instância de `BufferedImage` que contém a imagem de entrada e a instância de um objeto do tipo `photochopp.Filtro`, que especifica o filtro a ser aplicado. O método deve retornar uma nova instância de `BufferedImage` que conterá a imagem de saída.

f) A aplicação Java compilada deverá ser empacotada no arquivo "trabalho.jar" (veja as configurações de empacotamento disponíveis no NetBeans® em "Propriedades do Projeto\Construir\Empacotamento" e utilize o item de menu "Executar\Limpar e construir projeto" para criar o pacote).

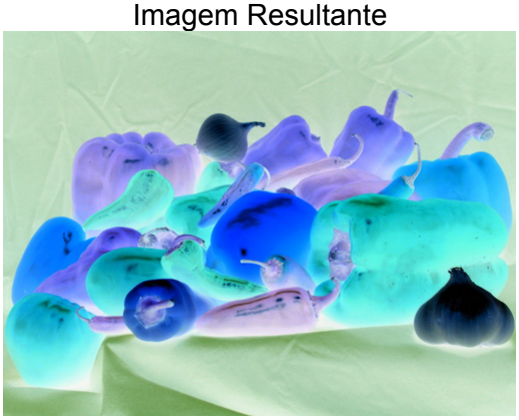
g) O usuário executará a aplicação única e exclusivamente através do prompt de comando do DOS ou Linux. Os argumentos esperados pela aplicação são listados a seguir.

`complemento "arquivo de entrada" "arquivo de saída"`

Converte a imagem de entrada colorida (espaço de cores RGB ou BGR) para sua imagem complementar de saída. Para tanto deverá ser aplicada a implementação da operação definida no item (b) sobre todos os pixels da imagem de entrada. Segue um exemplo de resultado esperado:

Imagem de Entrada

Imagem Resultante



`rgb-para-cinza "arquivo de entrada" "arquivo de saída"`

Converte a imagem de entrada colorida (espaço de cores RGB ou BGR) para uma imagem de saída em tons de cinza. Para tanto deverá ser aplicada a implementação da operação definida no item (c) sobre todos os pixels da imagem de entrada. Segue um exemplo de resultado esperado:

Imagem de Entrada

Imagem Resultante

`filtro-gaussiano "arquivo de entrada" "arquivo de saída"`

Aplica sobre cada canal da imagem de entrada colorida (espaço de cores RGB ou BGR) um filtro Gaussiano passa-baixa e salva o resultado em uma imagem colorida que utiliza o mesmo espaço de cores da entrada. O efeito esperado em um filtro passa-baixa é a remoção das altas frequências da imagem original, o que causa a suavização ou "borramento" da imagem. Para tanto deverá ser aplicada a implementação do operador de convolução discreta 2D definido no item (d) sobre a imagem de entrada f e o filtro w de tamanho 5×5 definido como:

0.0352	0.0387	0.0398	0.0387	0.0352
0.0387	0.0425	0.0438	0.0425	0.0387
0.0398	0.0438	0.0452	0.0438	0.0398
0.0387	0.0425	0.0438	0.0425	0.0387
0.0352	0.0387	0.0398	0.0387	0.0352

Segue um exemplo de resultado esperado:

Imagem de Entrada

Imagem Resultante

O filtro Gaussiano passa-baixa com os valores indicados acima deve ser codificado dentro da classe `photochopp.FiltroGaussiano`, uma subclasse de `photochopp.Filtro`.

`bordas limiar "arquivo de entrada" "arquivo de saída"`

Processa a imagem de entrada em tons de cinza e produz uma imagem de saída binária onde pixels brancos representam pixels de borda e pixels pretos representam pixels que não são de

borda. Esta operação pode ser implementada pela aplicação de duas convoluções sobre a imagem de entrada f. A primeira utiliza o filtro Sobel 3×3 horizontal, w_x:

-1.0	0.0	+1.0
-2.0	0.0	+2.0
-1.0	0.0	+1.0

O resultado é a imagem intermediária g_x. A segunda convolução aplica sobre a imagem de entrada o filtro Sobre 3×3 vertical, w_y:

-1.0	-2.0	-1.0
0.0	0.0	0.0
+1.0	+2.0	+1.0

O resultado desta convolução é a imagem intermediária g_y. A imagem final e é calculada a partir das duas imagens intermediárias por:

$$e(x,y) = \begin{cases} 255 & \text{caso } g(x,y) \geq t \\ 0 & \text{caso contrário} \end{cases}$$

onde

$$g(x,y) = \sqrt{\left(g_x(x,y)\right)^2 + \left(g_y(x,y)\right)^2}$$

e t é o valor real positivo informado pelo usuário no argumento `limiar` da chamada do programa. Segue um exemplo de resultado esperado:

Imagem de Entrada

Imagem Resultante

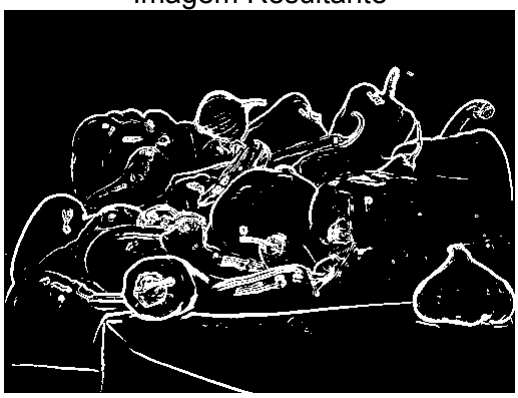


Imagem intermediária g_x

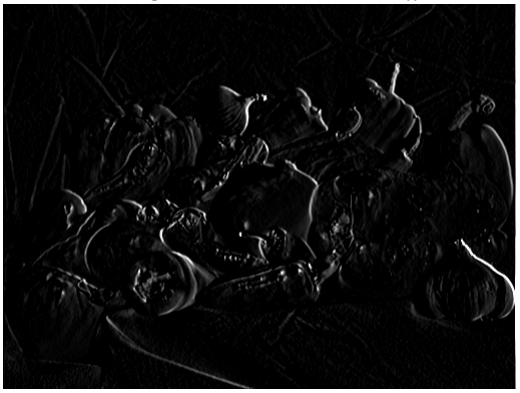
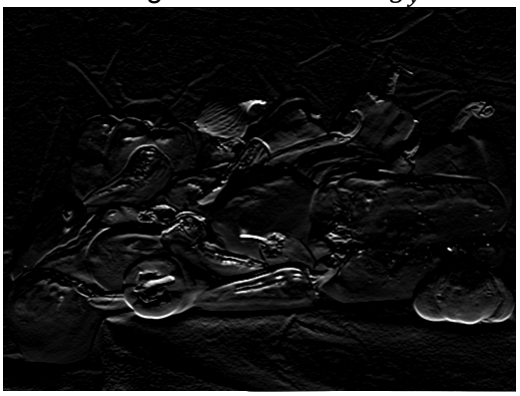


Imagem intermediária g_y



Os filtros de Sobel horizontal e vertical devem ser codificados dentro das classes `photochopp.FiltroSobelHorizontal` e `photochopp.FiltroSobelVertical`, respectivamente. Ambas subclasses de `photochopp.Filtro`.

A operação de detecção de bordas deverá ser implementada em um método estático da classe `photochopp.Operacoes`. Ele deverá receber como argumentos a instância de `BufferedImage` que contém a imagem de entrada e o valor de limiar. O método deverá retornar outra instância de `BufferedImage` que conterá a imagem de saída.

```
filtro-linear K w11 w12 ... w1K w21 w22 ... w2K ... wK1 wK2 ... wKK
              "caminho do arquivo de entrada" "caminho do arquivo de saída"
```

Aplica sobre cada canal da imagem de entrada colorida (espaço de cores RGB ou BGR) um filtro linear `K×K` definido pelo usuário e salva o resultado em uma imagem colorida que utiliza o mesmo espaço de cores da entrada. A disposição dos coeficientes reais `wij` do filtro obedece a seguinte convenção:

w11	w12	...	w1K
w21	w22	...	w2K
⋮	⋮		⋮
wK1	wK2	...	wKK

Obs. 1: As operações aplicadas sobre as imagens deverão estar implementadas dentro no pacote `photochopp`. Ou seja, não implemente nenhuma operação de processamento ou composição de processamentos dentro no método `main` de sua classe principal.

Obs. 2: Utilize de forma adequada modificadores de visibilidade.

Obs. 3: Levante e trate exceções de forma adequada, tanto nos métodos implementados dentro das classes do pacote `photochopp` (e.g., métodos que esperam uma instância de objeto como argumento, mas que recebem uma referência nula) quanto no método `main` (e.g., argumentos inválidos informados pelo usuário).

Obs. 4: Variações na forma de interação por argumentos não serão toleradas.

Bom Trabalho!