Programação de Software Básico - Trabalho 1

Introdução

O primeiro trabalho da disciplina consiste em adicionar novas funções à uma biblioteca para o processamento de imagens. Para isso, deverá ser criada uma aplicação de demonstração que utiliza pelo menos 3 imagens como entrada, aplica diferentes filtros e gera diversas imagens como saída, apresentando a funcionalidade de cada um dos filtros implementados em cada uma das imagens.

As imagens de entrada deverão ser uma coloridas com precisão de 24 bits por pixel (R, G, B), onde cada componente possui um valor de intensidade de 0 a 255. As imagens utilizadas como entrada deverão possuir tamanhos variados (acima de 512x512 pixels) e no máximo uma das imagens poderá ser quadrada. Para a manipulação de imagens, está sendo fornecida uma biblioteca que permite ler e escrever imagens no formato PPM (*Portable Pix Map*).

Leitura e escrita de imagens do tipo PPM

Uma imagem pode ser representada por uma matriz de pixels onde cada cor é definida por 3 componentes: vermelho (R), verde (G) e azul (B). Cada uma dessas componentes usualmente é codificada em um byte, o que produz 3 bytes por pixel (24 bits) - ou seja, 16 milhões de possíveis cores. Em outras palavras, as intensidades (R, G, B) variam de 0 a 255, onde 0 é escuro e 255 é claro.

FFFFFF	000000	333333	666666	999999	CCCCCC	CCCC99	9999CC	666699
660000	663300	996633	003300	003333	003399	000066	330066	660066
990000	993300	CC9900	006600	336666	0033FF	000099	660099	990066
CC0000	CC3300	FFCC00	009900	006666	0066FF	0000CC	663399	CC0099
FF0000	FF3300	FFFF00	00CC00	009999	0099FF	0000FF	9900CC	FF0099
CC3333	FF6600	FFFF33	00FF00	00CCCC	00CCFF	3366FF	9933FF	FF00FF
FF6666	FF6633	FFFF66	66FF66	66CCCC	00FFFF	3399FF	9966FF	FF66FF
FF9999	FF9966	FFFF99	99FF99	66FFCC	99FFFF	66CCFF	9999FF	FF99FF
FFCCCC	FFCC99	FFFFCC	CCFFCC	99FFCC	CCFFFF	99CCFF	CCCCFF	FFCCFF

Veja como diversas cores são representadas nesse formato - cada cor está expressa pelas componentes RGB em hexadecimal.

O código fornecido define duas *structs*: uma para representar um pixel e outra para representar a imagem inteira. Após a leitura da imagem, os pixels estarão disponíveis a partir do ponteiro *image*.

```
/* lib_ppm.h */
struct pixel_s {
    unsigned char r, g, b;
};

struct image_s {
    int width;
    int height;
    struct pixel_s *pix;
};

int read_ppm(char *file, struct image_s *image);
int write_ppm(char *file, struct image_s *image);
int new_ppm(struct image_s *image, int width, int height);
int free_ppm(struct image_s *image);

/* declaração de uma estrutura de dados para imagem */
struct image_s data;
struct image_s *image = &data;

/* leitura da imagem */
r = read_ppm("lena.ppm", image);
...
/* modificando o valor do pixel [20, 50] (posição X, Y) */
image->pix[50 * image->width + 20].r = 255;
image->pix[50 * image->width + 20].g = 255;
image->pix[50 * image->width + 20].b = 255;
...
/* escrita da imagem */
write_ppm("copy.ppm", image);
```

Definição dos filtros a serem implementados

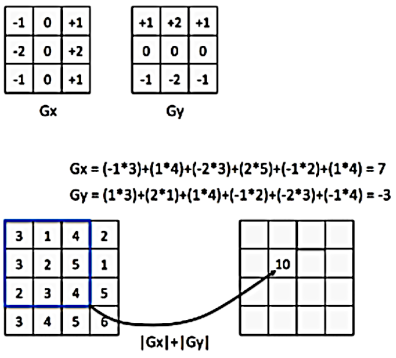
Os seguintes filtros para o processamento de imagens deverão ser implementados:

1. Imagem em tons de cinza - para converter uma imagem para tons de cinza deve-se extrair a componente luminância, calculado com base em valores ponderados dos canais R, G e B de cada pixel por meio da fórmula $Y = R * 0.299 + G * 0.587 + B * 0.114$. Deve-se copiar o mesmo resultado de cada pixel para todas as componentes da imagem destino.

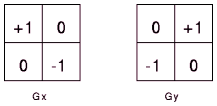
2. Operação *threshold* - inicialmente, deve-se converter a imagem para tons de cinza. Posteriormente, para cada pixel da imagem, deve-se verificar se o seu valor está abaixo do limiar (threshold) definido pelo

usuário. Se estiver, o valor do pixel resultante será 0 (preto), caso contrário será 255 (branco). Deve-se copiar o mesmo resultado para todas as componentes da imagem destino.

3. Filtro Sobel - é um filtro utilizado para a detecção de bordas em uma imagem, onde é utilizado um operador de convolução com dimensão 3x3. Nesse filtro, em uma versão simplificada ignora-se os pixels da periferia da imagem resultante (não recebem valor). Cada pixel da imagem resultante é calculado por meio da aplicação de duas matrizes (vertical e horizontal) que são convoluídas com a pixels da imagem original. O pixel gerado a partir da aplicação do operador é o central a partir das matrizes. O valor final do pixel é dado por $\sqrt{Gx^2 + Gy^2}$, e pode ser aproximado por $|Gx| + |Gy|$. Esse filtro deve ser aplicado independentemente em cada componente da imagem.



4. Filtro Roberts Cross - assim como o filtro anterior, esse pode ser utilizado para detecção de bordas em uma imagem, sendo utilizado um operador com dimensão 2x2. O funcionamento é idêntico ao filtro anterior, com diferença apenas na dimensão e valores dos operadores de convolução. O pixel gerado a partir da aplicação do operador é o superior esquerdo a partir das matrizes. O valor final do pixel é dado por $\sqrt{Gx^2 + Gy^2}$, e pode ser aproximado por $|Gx| + |Gy|$. Esse filtro deve ser aplicado independentemente em cada componente da imagem.



Para cada filtro, deve ser implementada a API apresentada a seguir. Essa API deverá ser disponibilizada à aplicação de demonstração por meio de um

arquivo de cabeçalho (.h) e a implementação deverá ser definida em um código fonte (.c). Os códigos fonte deverão ser compilados individualmente e farão parte de uma biblioteca estática (.a), a ser criada pelo grupo. A aplicação de demonstração não deve incluir nada a não ser o cabeçalho contendo os protótipos da API e a lógica para demonstração de uso dos filtros, sendo a resolução dos símbolos (referências às funções da API) definida em tempo de ligação com a biblioteca estática. O Makefile deve ser capaz de compilar a biblioteca e a aplicação de demonstração individualmente.

```
int grayscale(struct image_s *src_image, struct image_s *dst_image);
int threshold(struct image_s *src_image, struct image_s *dst_image, unsigned
    char value);
int sobel(struct image_s *src_image, struct image_s *dst_image);
int roberts(struct image_s *src_image, struct image_s *dst_image);
```

Para aplicação de um filtro, deve-se carregar uma imagem (função *read_ppm()*) e criar um canvas em branco, com as dimensões da imagem original (função *new_ppm()*). No canvas será armazenado o resultado do processamento, que poderá ser gravado no disco (função *write_ppm()*).

Avaliação

Este trabalho deverá ser realizado em grupos de dois ou três integrantes e apresentado em sala de aula na data indicada no Moodle (apresentação em torno de 10 minutos). Para a entrega, é esperado que apenas um dos integrantes envie pelo Moodle, até a data e hora especificadas, um arquivo *.tar.gz* ou *.zip* do projeto contendo o código fonte desenvolvido.

Observações

- O código deve estar indentado corretamente (qualquer IDE de programação decente faz isso automaticamente).
- A cópia parcial ou completa do trabalho terá como conseqüência a atribuição de nota ZERO ao trabalho dos alunos envolvidos. A verificação de cópias é feita inclusive entre turmas.
- A cópia de código ou algoritmos existentes da Internet também não é permitida. Se alguma ideia encontrada na rede for utilizada na implementação, a referência deve constar no código em um comentário.