Projeto 1 - processamento de imagens

Igor Rafael Hashi

21-02-2018

No primeiro projeto vamos trabalhar com imagens! Iremos fazer três programas que fazem três processamentos de imagens diferentes e que usam funções auxiliares para ler e escrever imagens do disco.

Parte 1: Entrada e saída com imagens

Vamos ler e escrever imagens no formato *PGM*. Este formato de imagens aceita somente imagens em níveis de cinza e é muito simples de programar. Cada arquivo tem um cabeçalho com o seguinte formato.

```
P2
W H
V
```

Onde W e H são a largura e a altura da imagem e V é o valor equivalente ao branco. Nos nossos exemplo `V=255`. Em seguida temos os `W*H` valores dos pixels impressos em texto puro. Todos estes valores devem estar entre `0` e `255`.

Tarefa 1: visualize o arquivo entrada1.pgm usando um editor de texto.

Vamos representar nossa imagem usando a seguinte estrutura:

```
typedef struct {
    int w, h;
    unsigned char pixels[512][512];
} image_t;
```

Tarefa 2: crie um arquivo `image_io.c` (e seu cabeçalho correspondente `image_io.h` e programe as seguintes funções:

- `void image_read(char *path, image_t*out);`
- `void image_write(char *path, image_t*in);`

A função `image_read` retorna seu resultado em um pointeiro para `image_t`.

Tarefa 3: crie um arquivo `rename_image.c` que recebe dois argumentos na linha de comando. Seu programa deve ler a imagem passada no primeiro argumento e salvar ela com o nome passado no segundo argumento.

```
> $ ./rename_image entrada1.pgm entrada1-copia.pgm
```

Vamos agora trabalhar com processamento de imagens! O primeiro processamento que faremos é o limiar (que já trabalhamos no primeiro dia).

$$out(y,x) = \begin{cases} 255 & \text{se } in(y,x) > l \\ 0 & \text{c.c} \end{cases}$$

Tarefa 4: crie um programa *limiar.c* que recebe três argumentos na linha de comando: imagem original, imagem destino e um inteiro l. Seu programa deve executar o processamento acima e escrever o resultado na imagem destino.

Para organizar melhor o código, implementa o limiar em uma função com o seguinte protótipo:

```
void limiar(image_t *in, image_t *out, int lim);
```

Figure 1: Resultado para entrada5.pgm com lim=127

Agora vamos gerenciar nosso projeto usando *Make*.

Tarefa 5: crie um *Makefile* para compilar ambos os executáveis. Não se esqueça de adicionar a regra `all` para compilar todos os executáveis ao executar `make` sem parâmetros.

Mais detalhes sobre `make all`: https://stackoverflow.com/questions/2514903/what-does-all-stand-for-in-a-makefile

---

O segundo processamento que implementaremos é a normalização de contraste (também chamado de *contrast stretching*).

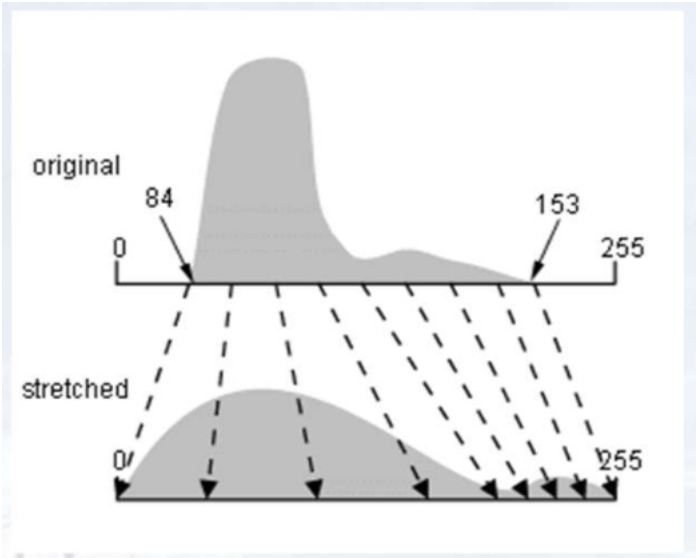

Figure 2: fonte: https://stackoverflow.com/questions/41118808/difference-between-contrast-stretching-and-histogram-equalization

Este processamento mapeia o menor nível de cinza para 0 e o maior para 255, esticando tudo o que está no meio de maneira linear.

Tarefa 6: cria um programa `contrast.c` que recebe dois argumentos na linha de comando. Seu programa deve ler a imagem passada no primeiro argumento, aplicar a transformaçào de contraste e salvar o resultado no segundo argumento.

Não se esqueça de modificar seu *Makefile* de acordo.

Para organizar melhor o código, implementa o este processamento em uma função com o seguinte protótipo:

```
void contrast(image_t *in, image_t *out);
```

Nosso último processamento será a convolução. Dada uma matriz k 3×3, posicionamos esta matriz em cima de cada pixel da imagem e fazemos a média ponderada dos pixels vizinhos usando os valores de k como pesos.

Exemplo: dada a matriz k definida abaixo,

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Figure 3: Resultado para a entrada6.pgm

devemos fazer o seguinte processando

$$\begin{aligned} out(y,x) = &+0 \times in(y-1,x-1) \quad -1 \times in(y-1,x) + 0 \times in(y-1,x+1) \\ &-1 \times in(y,x-1) \qquad\quad +4 \times in(y,x) - 1 \times in(y,x+1) \\ &+0 \times in(y+1,x-1) \quad -1 \times in(y+1,x) + 0 \times in(y+1,x+1) \end{aligned}$$

Dica: seu código ficará mais limpo se você computar a equação acima usando um loop ;)

Tarefa 7: faça uma função `void convolucao(image_t *in, image_t*out, double kernel[3][3]);` que executa a convolução para uma matriz k fornecida pelo usuário. Esta função deve estar em um arquivo separado, pois será usada nas próximas duas tarefas.

Tarefa 8: faça um programa `blur.c` que recebe dois nomes de imagens na linha de comando e aplica a convolução feita na função acima com a seguinte máscara.

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

Figure 4: Resultado do blur na entrada5.pgm

Tarefa 9: faça um programa `borders.c` que recebe dois nomes de imagens na linha de comando e aplica a convolução feita na função acima com a seguinte máscara.

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Figure 5: Resultado para entrada7.pgm