

Arquitectura de Sistemas e Computadores I

Licenciatura em Engenharia Informática

Miguel Barão

Ano Lectivo 2016-2017

1 Objectivo

Pretende-se com este trabalho desenvolver um conjunto de funções em assembly MIPS para fazer detecção de contornos em imagens a cores. Dado um ficheiro com uma imagem a cores RGB, o programa deverá gerar outro ficheiro de imagem, em tons de cinzento, com fundo branco e com traços escuros nos locais onde existem contornos na imagem original (ver figuras 1(a) e 1(e)).

Para efectuar a detecção de contornos é necessário efectuar uma sequência de passos:

1. Leitura do ficheiro com a imagem em formato RGB;
2. Conversão da imagem de RGB para escala de cinzentos (*gray scale*);
3. Aplicação de um operador Sobel vertical para a detecção de variações verticais na intensidade da cor;
4. Aplicação de um operador Sobel horizontal para a detecção de variações horizontais na intensidade da cor;
5. Combinação das variações de cor nas direcções horizontal e vertical para a obtenção do resultado final;
6. Escrita da imagem final num ficheiro em formato GRAY.

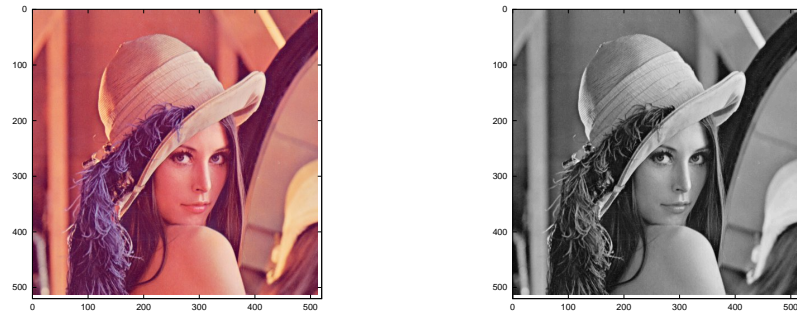
As imagens da figura 1 ilustram alguns destes passos. As secções seguintes explicam detalhadamente como é efectuado cada um destes passos.

2 Teoria

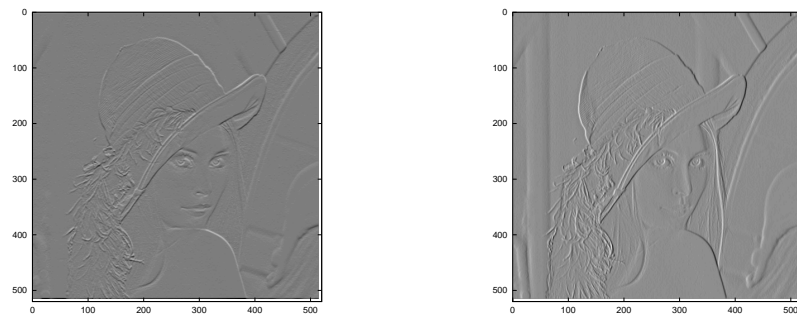
2.1 Cores RGB

Existe uma infinidade de cores distintas, mas o olho humano é genericamente sensível a apenas três cores: vermelho, verde e azul. Assim, quando o objectivo é gerar uma fotografia para ser observada pelo olho humano basta representar essas três cores. Quando falamos em RGB referimo-nos simplesmente à representação de uma cor misturando as três componentes *Red*, *Green* e *Blue* com várias intensidades.

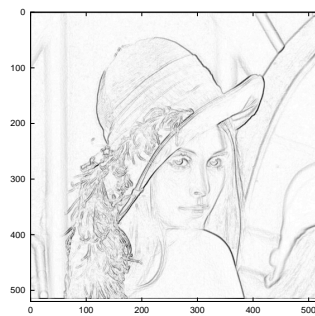
Num computador, cada uma das três intensidades é representada por um número inteiro com N bits. Se $N = 8$ bits (1 byte), então temos níveis de intensidade de 0 a 255 para cada componente de cor. Como existem $256^3 = 16777216$ combinações diferentes, temos mais de 16 milhões de cores diferentes.



(a) Imagem “Lena” original, tamanho 512×512 . (b) Imagem “Lena” convertida para escala de cinzentos.



(c) Imagem filtrada pelo operador Sobel vertical. (d) Imagem filtrada pelo operador Sobel horizontal.



(e) Contornos obtidos a partir da aplicação dos operadores Sobel horizontal e vertical.

Figura 1: Imagem de teste “Lena” e passos intermédios para a detecção de contornos.

2.2 Formato de imagem RGB

O ficheiro de imagem original contém uma imagem a cores RGB, onde a intensidade de cada cor é codificada por um byte com valores de 0 a 255. Assim, cada pixel é codificado por três bytes, um byte para cada uma das cores primárias vermelho (R), verde (G) e azul (B).

Uma imagem completa consiste numa matriz em que cada elemento é um pixel de cor RGB. Uma imagem de dimensão $m \times n$ corresponde à matriz

$$\begin{bmatrix} (R_{11}G_{11}B_{11}) & (R_{12}G_{12}B_{12}) & \cdots & (R_{1n}G_{1n}B_{1n}) \\ (R_{21}G_{21}B_{21}) & (R_{22}G_{22}B_{22}) & \cdots & (R_{2n}G_{2n}B_{2n}) \\ \vdots & \vdots & & \vdots \\ (R_{m1}G_{m1}B_{m1}) & (R_{m2}G_{m2}B_{m2}) & \cdots & (R_{mn}G_{mn}B_{mn}) \end{bmatrix}.$$

Os pixels da imagem anterior são guardados sequencialmente num ficheiro pela seguinte ordem:

$$\left[\underbrace{R_{11}G_{11}B_{11} \cdots R_{1n}G_{1n}B_{1n}}_{1^{\text{a}} \text{ linha}} \underbrace{R_{21}G_{21}B_{21} \cdots R_{2n}G_{2n}B_{2n}}_{2^{\text{a}} \text{ linha}} \cdots \underbrace{R_{m1}G_{m1}B_{m1} \cdots R_{mn}G_{mn}B_{mn}}_{m\text{-ésima linha}} \right]$$

O ficheiro RGB não contém informação acerca do tamanho da imagem, nem o número de bits usado para codificar cada cor. Essa informação é perdida na conversão. Para descodificar uma imagem neste formato é necessário conhecer de antemão as características da imagem original.

Para converter uma imagem para este formato, pode ser usado o programa `convert` disponibilizado pelo `ImageMagick` em Linux. Ilustra-se em seguida a utilização deste programa na conversão entre os formatos JPEG, RGB e GRAY (escala de cinzentos).

```
convert imagem.jpg imagem.rgb          # JPG  -> RGB
convert -size 512x512 -depth 8 imagem.rgb imagem.jpg    # RGB  -> JPG
convert -size 512x512 -depth 8 imagem.gray imagem.jpg    # GRAY -> JPG
```

No segundo caso, correspondente à conversão de RGB para JPEG, e como o formato RGB não guarda a dimensão da imagem, é necessário passar como opções a dimensão `-size 512x512` e o número de bits de cada cor `-depth 8`. O terceiro exemplo ilustra a conversão de uma imagem monocromática (em escala de cinzentos) para JPEG. Neste último caso cada pixel é codificado por um byte com valores de 0 a 255 (preto até branco).

2.3 Conversão RGB para tons de cinzento

A conversão RGB para tons de cinzento consiste em converter as três componentes RGB para apenas uma componente que codifica a intensidade luminosa total de um pixel.

Sabe-se que a retina humana tem sensibilidades diferentes para cores diferentes, sendo mais sensível ao verde e menos ao azul. Assim, se se pretender que a imagem em tons de cinzento reflecta estas diferenças de sensibilidade, as cores têm pesos diferentes na intensidade luminosa de um pixel. A fórmula seguinte é um *standard* neste tipo de conversão:

$$I = 0.30R + 0.59G + 0.11B$$

Cada pixel da imagem em tons de cinzento é obtido pela aplicação da fórmula anterior ao pixel correspondente da imagem RGB.

2.4 Operadores Sobel

Os operadores Sobel são usados para detectar variações de cor em imagens em tons de cinzento.

Um operador Sobel consiste numa matriz de dimensão 3×3 que é convolvida com a imagem original. São usadas duas matrizes:

$$S_h = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad S_v = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Cada um destes operadores vai detectar variações na intensidade de cor numa das direcções horizontal ou vertical.

A aplicação destes operadores é efectuada por uma operação chamada convolução 2D, e que se exemplifica em seguida. Suponha-se que a imagem original (em escala de cinzentos) está representada numa matriz A . A Imagem B , obtida pela convolução de A com o operador S_h ,

$$B = S_h * A,$$

onde o símbolo $*$ denota a operação de convolução, é obtida da seguinte maneira: percorre-se cada posição (i, j) da imagem B , e calcula-se

$$B(i, j) = \sum_{p \in \{-1, 0, 1\}} \sum_{q \in \{-1, 0, 1\}} A(i + p, j + q) S_h(2 - p, 2 - q).$$

Esta operação consiste essencialmente em sobrepôr a matriz S_h centrada na componente (i, j) da matriz A , fazer todos os produtos de elementos de S_h com os respectivos de A , e somar tudo obtendo-se no final $B(i, j)$. Este procedimento é repetido para calcular todos os elementos de B .

A convolução não está bem definida junto às margens da imagem A . Nessas regiões a solução fica ao critério do aluno.

2.5 Detecção de contornos

A detecção dos contornos é efectuada combinando os resultados obtidos com a aplicação dos operadores Sobel S_h e S_v à imagem A . Suponha-se que se obteve as matrizes

$$B_h = \frac{1}{4} |S_h * A|$$

$$B_v = \frac{1}{4} |S_v * A|$$

onde as barras $|\cdot|$ denotam o valor absoluto aplicado a cada elemento da matriz.

A matriz C com os contornos pode ser obtida somando as matrizes B_h e B_v anteriores:

$$C = \frac{1}{2} (B_h + B_v).$$

A imagem que se obtém tem intensidade baixa (cor escura) quando não há variações de contorno e alta (cor clara) nos contornos. Para obter um resultado como o da figura 1(e) é necessário determinar a imagem complementar:

$$D(i, j) = 255 - C(i, j)$$

para todos os pixels (i, j) .

3 Implementação

A implementação da detecção de contornos em assembly MIPS deve ser estruturada em várias funções independentes. As funções a desenvolver são as seguintes:

read_rgb_image função que lê uma imagem no formato RGB para um array em memória.

A função recebe como argumentos uma string com o nome do ficheiro a ler, e devolve um buffer com a imagem RGB lida. Assume-se dimensão 512×512 .

write_gray_image função que escreve uma imagem em formato GRAY num ficheiro. A função recebe como argumentos o nome de um ficheiro, um buffer com a imagem e o comprimento do buffer (por esta ordem).

rgb_to_gray função que converte uma imagem a cores RGB para uma imagem em tons de cinzento GRAY. A função recebe como argumentos um buffer com a imagem RGB e um buffer onde deve ser colocada a imagem em formato GRAY. A função assume internamente que a imagem tem dimensão 512×512 .

convolution função que calcula a convolução de uma imagem A com um operador Sobel (matriz 3×3) e coloca o resultado numa matriz B . Os valores da matriz B devem estar em valor absoluto. A função recebe como argumentos um buffer com a matriz A , um buffer com um dos operadores Sobel e um buffer que vai conter a imagem filtrada B . A função assume internamente que a imagem tem dimensão 512×512 .

contour função que calcula a imagem final combinando duas imagens filtradas. A função recebe como argumentos dois buffers com as imagens a combinar, e um buffer que vai conter o resultado. A função assume internamente que a imagem tem dimensão 512×512 .

As funções assumem internamente um tamanho de imagem fixo de 512×512 . Se quiser usar dimensões diferentes pode alterar as funções de modo a receber o tamanho da imagem como argumento.

3.1 I/O para Ficheiros

A leitura e escrita de ficheiros é efectuada da mesma maneira que em sistemas UNIX/LINUX com as funções `open`, `read`, `write` e `close`.

Para abrir um ficheiro para leitura pode usar-se o código seguinte:

```
la $a0, FILE          # string containing filename to open
li $a1, 0
li $a2, 0              # read only
li $v0, 13             # open
syscall                # v0 = file descriptor
```

No final, o registo `$v0` contém o descritor do ficheiro que será usado para as operações de leitura seguintes. A leitura do ficheiro é efectuada com um `read` da seguinte maneira:

```
# a0 = file descriptor
# a1 = buffer address
# a2 = length (number of bytes to read)
li $v0, 14             # read
syscall
```

Para abrir um ficheiro para escrita pode usar-se o código seguinte:

```
la $a0, FILE          # string containing filename to open
li $a1, 577            # flags O_CREAT|O_TRUNC|O_WRONLY
li $a2, 0x1ff          # write permissions 777
li $v0, 13             # open
syscall                # v0 = file descriptor (-1 on failure)
```

A escrita neste ficheiro é efectuada com um `write` da seguinte maneira:

```
# a0 = file descriptor
# a1 = buffer address
# a2 = length (number of bytes to write)
li $v0, 15             # write
syscall
```

No final todos os ficheiros abertos devem ser fechados com a função `close`:

```
# a0 = file descriptor
li $v0, 16          # close
syscall
```

Atenção: O manual do SPIM que vem com algumas edições do livro tem informação errada no que respeita a estes syscalls. Ver documentação do Mars no menu “*help*”.

4 Notas finais

- O trabalho deverá ser realizado em grupos de 2 alunos.
- Comece com imagens de muito baixa dimensão para testar o código. Por exemplo uma imagem 4×4 com um total de 16 pixeis. Quando tudo funcionar correctamente numa imagem pequena, experimente a imagem lena com dimensão 64×64 .
- Poderá ser útil usar as instruções `madd`, `maddu`, `sra` e `lbu`, e a pseudo-instrução `abs`.
- O código deve estar comentado do seguinte modo:

```
#####
# xpto - Esta funcao calcula a area de um retangulo
#
# Argumentos:
#  a0 - largura
#  a1 - comprimento
#
# Retorna:
#  v0 - area calculada
#####
xpto:
    mult $a0, $a1    # calcula area
    ...
    jr $ra
    nop
```

- Além do código devidamente comentado e bem organizado, será necessário entregar um relatório em PDF onde explica o problema e como organizou a sua solução, mas sem incluir código. (Sugestão: use \LaTeX para a escrita do relatório)
- Submeter o trabalho no moodle num único ficheiro comprimido (rar, zip, tar.gz, etc) com nome formado pelos números dos alunos, por exemplo `11111-22222.rar`.
- O trabalho é longo e deverá demorar várias semanas a realizar. Guarde a última semana exclusivamente para a escrita do relatório.

Bom trabalho!
Miguel Barão