u évora
ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Enunciado do Trabalho de

Arquitectura de Computadores I

Licenciatura em Engenharia Informática

2022–2023

1 Objectivo

Pretende-se desenvolver um programa em Assembly RISC-V para localizar personagens da saga StarWars numa imagem. Dado um ficheiro com uma imagem no formato RGB, o programa deverá gerar uma nova imagem que identifique a personagem escolhida pelo utilizador.

Figura 1: Yoda, Darth Maul e Mandalorian.

2 Método

Observando a imagem da figura 1, observa-se que cada personagem tem uma tonalidade de cor diferente. O mestre Yoda tem tons mais esverdeados, o Darth Maul é mais vermelho e o Mandalorian tem uma tonalidade mais perto do ciano. Como estes personagens são distinguiveis apenas pela sua tonalidade, basta localizar as regiões onde cada tonalidade é mais dominante.

Para localizar um personagem na imagem, procuram-se os pixeis com uma gama de tonalidades correspondente ao personagem e depois calcula-se o "centro de massa" (posição média) dos pixeis com essa tonalidade. A posição encontrada fica aproximadamente no centro do personagem. Sabendo as coordenadas, pode-se então extrair, seleccionar ou marcar a personagem.

3 Espaço de cores RGB

As imagens que vemos são consequência da luz refletida pelos objetos que atinge os fotoreceptores existentes na retina dos nossos olhos. O sistema visual humano é sensível a três cores primárias: vermelho, verde e azul (*RGB — Red/Green/Blue*). Outros animais podem ser sensíveis a mais ou menos cores.

Na representação de imagens em computador, o destinatário é normalmente humano. Por este motivo, a tecnologia desenvolveu-se de maneira a adaptar-se especificamente aos humanos. Equipamentos tais como monitores, câmaras fotográficas e de vídeo suportam tipicamente apenas as três cores primárias que os humanos têm capacidade de ver.

A representação de cores em RGB refere-se à representação de uma cor somando as três componentes *Red*, *Green* e *Blue* com intensidades luminosas apropriadas.

Num computador, cada uma das três intensidades RGB é normalmente representada por um número com N bits. O número de bits usado chama-se *profundidade de cor* (*color depth*).

Uma das profundidades de cor muito usadas, chamada *True color*, usa 8 bits para cada componente de cor RGB obtendo-se no total 24 bits por pixel. Neste caso, a intensidade de cada componente é um número inteiro de 0 a 255:

- o vermelho puro é representado por $(255, 0, 0)$;
- o branco é a mistura das três componentes RGB: $(255, 255, 255)$;
- o preto não é bem uma cor, mas sim a ausência de luz: $(0, 0, 0)$.

Em *True color* existem $256 \times 256 \times 256 = 16777216$ combinações diferentes RGB e portanto temos cerca de 16 milhões de cores distintas.

4 Formato de imagem RGB

Neste trabalho, o ficheiro de imagem original contém uma imagem a cores no formato RGB com 8 bits de profundidade de cor. Assim, cada componente de cor é codificada por um byte e portanto cada pixel tem três bytes.

Uma imagem completa consiste numa matriz em que cada elemento é um pixel. 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 pixeis da imagem são guardados sequencialmente num ficheiro pela ordem *raw major*[1]:

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

[1] A ordenação em que os elementos de cada linha são guardados sequêncialmente juntos uns aos outros chama-se *raw major*. A outra possibilidade é *column major* em que a matriz é guardada por colunas. https://en.wikipedia.org/wiki/Row-_and_column-major_order

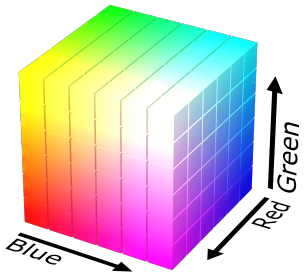


Figura 2: Cubo de cores RGB.

O ficheiro RGB não contém informação acerca do tamanho da imagem nem da profundidade de cor. Para descodificar uma imagem neste formato é necessário conhecer essas propriedades.

No terminal, o comando `convert` permite converter entre diversos formatos de imagem. Alguns exemplos de conversão entre os formatos JPEG, PNG e RGB são os seguintes:

```
convert imagem.jpg imagem.rgb                          # JPG -> RGB
convert -size 320x180 -depth 8 imagem.rgb imagem.png   # RGB -> PNG
convert -size 320x180 -depth 8 imagem.rgb imagem.jpg   # RGB -> JPG
```

Como o formato RGB não inclui as dimensões da imagem nem a profundidade de cor, para converter a partir de RGB é necessário dar os parâmetros `-size 320x180` e `-depth 8`.

O comando `convert` é disponibilizado pelo pacote `ImageMagick` em Linux.

5 Espaço de cores HSV

O espaço de cores HSV codifica as cores nas componentes *Hue*, *Saturation* e *Value*:

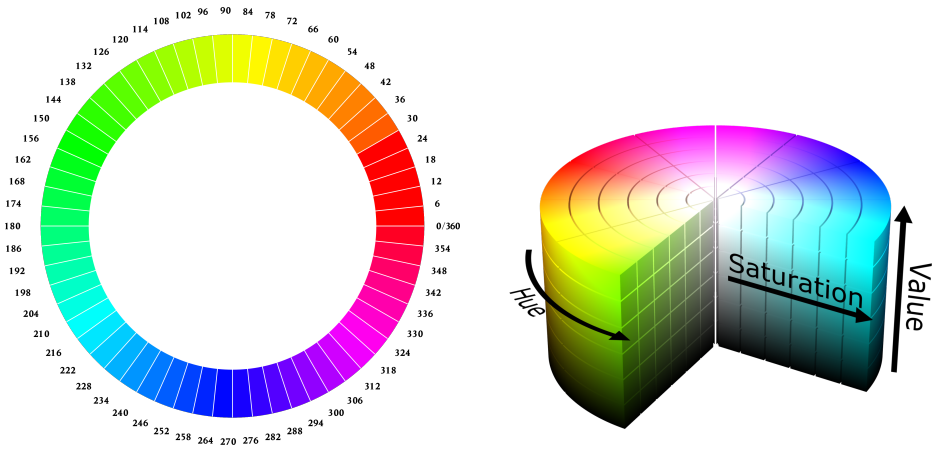
Hue representa a tonalidade num círculo de cores.

Saturation representa a pureza da cor. Uma cor pura diz-se saturada. Misturando branco, a cor esbate-se e é menos saturada. O cinzento tem saturação zero.

Value representa a iluminação. O valor zero representa a escuridão, ou seja o preto. Um valor alto significa uma boa iluminação e as cores são bem visíveis.

Enquanto que em RGB a mesma tonalidade tem valores diferentes consoante a iluminação ambiente, em HSV a tonalidade é codificada numa única componente *Hue* que não varia consoante a iluminação. Por este motivo, o HSV é mais apropriado do que o RGB para detetar tonalidades de cor.

Tanto o espaço de cores RGB como o HSV são espaços tridimensionais, mas com sistemas de coordenadas diferentes. As cores em RGB são representadas num cubo em que cada dimensão corresponde a uma componente de cor, como na figura 2. No caso do HSV as cores são representadas num cilindro. Isto acontece porque a componente Hue é circular: dando uma volta regressa-se à mesma cor, como ilustra a figura 3(a). A figura 3(b) mostra todo o espaço de coordenadas no cilindro HSV.



(a) Círculo de cores Hue

(b) Cilindro HSV

Figura 3: Espaço de cores HSV.

Secção	Nome	Hue (°)
$R > G \geq B$	Laranja	$60\frac{G-B}{R-B}$
$G \geq R > B$	Verde-amarelado	$120 - 60\frac{R-B}{G-B}$
$G > B \geq R$	Verde primavera	$120 + 60\frac{B-R}{G-R}$
$B \geq G > R$	Azure	$240 - 60\frac{G-R}{B-R}$
$B > R \geq G$	Violeta	$240 + 60\frac{R-G}{B-G}$
$R \geq B > G$	Rosa	$360 - 60\frac{B-G}{R-G}$

Tabela 1: Cálculo da componente Hue.

A componente Hue pode ser calculada de forma aproximada pela tabela 1, resultando num valor inteiro entre 0 e 359 graus. Note que, no caso das três componentes RGB serem iguais, o Hue não está definido (terá de decidir o que fazer nesse caso).

Não se mostram aqui as fórmulas de cálculo das outras duas componentes S e V porque não serão necessárias neste trabalho.

6 Segmentação de imagem

O primeiro passo para localização dos personagens consiste em identificar os pixeis que pertencem a cada um. Este processo chama-se *segmentação da imagem*. Como cada personagem tem tonalidades próprias diferentes dos restantes, podemos distingui-los apenas pela componente Hue. A tabela 2 mostra os intervalos de Hue que se podem usar na segmentação.

As imagens da figura 4 mostram o resultado da segmentação para cada um dos personagens. Foram mantidos os pixeis de cada personagem e colocados a preto os restantes. O resultado não é perfeito, mas verifica-se que a grande maioria dos pixeis que permaneceram pertencem ao personagem pretendido.

Personagem	Hue
Yoda	$[40°, 80°]$
Darth Maul	$[1°, 15°]$
Mandalorian	$[160°, 180°]$

Tabela 2: Intervalos de Hue correspondentes a cada personagem.

(a) Yoda (b) Darth Maul (c) Mandalorian

Figura 4: Segmentação de imagem.

7 Localização dos personagens

De seguida, podemos calcular o "centro de massa" de cada personagem com as fórmulas:

$$c_x = \frac{1}{N}\sum_x\sum_y \mathrm{I}_p(x,y)x, \qquad c_y = \frac{1}{N}\sum_x\sum_y \mathrm{I}_p(x,y)y, \qquad N = \sum_x\sum_y \mathrm{I}_p(x,y).$$

Nestas fórmulas, x e y vão percorrer, respetivamente, todas as linhas e colunas da imagem. A função $\mathrm{I}_p(x,y)$ é uma função indicadora que, para cada personagem p, indica se o pixel de coordenadas (x,y) pertence ou não a esse personagem — 1 se pertence e 0 se não pertence — e N é o total de pixeis do personagem. As coordenadas (c_x, c_y) correspondem ao "centro de massa".

As fórmulas são simplesmente a média das coordenadas que pertencem a um personagem. Estes cálculos deverão ser feitos em aritmética inteira (não use instruções de vírgula flutuante).

8 Output

Após a localização do personagem ter sido obtida, o output do programa é sempre um novo ficheiro de imagem no formato RGB. A imagem a gerar pode ser uma das seguintes, à sua escolha:

- Imagem igual à original com uma mira em forma de cruz (+) sobreposta no centro de massa. Esta mira deve usar cores vermelha e verde puras para que seja facilmente visível sobre qualquer personagem.
- Imagem igual à original com um quadrado centrado no centro de massa. O tamanho do quadrado deve depender do número de pixeis N identificados com o personagem. Desta maneira, se um personagem for maior, então o respetivo quadrado também será maior.

- Semelhante ao anterior, mas em vez de desenhar um quadrado, produz uma imagem desse tamanho com a cara do personagem identificado. (terá também de mostrar na consola a dimensão da imagem para se poder converter de RGB para outro formato)

9 Implementação

A implementação deve estar estruturada em várias funções. Recomenda-se que implemente as seguintes funções:

- `read_rgb_image` lê um ficheiro com uma imagem no formato RGB para um array em memória. A função tem como parâmetros uma string com o nome do ficheiro a ler e o endereço do array onde a imagem deverá ser escrita.
- `write_rgb_image` cria um ficheiro novo com uma imagem no formato RGB.
- `hue` calcula a componente Hue a partir das componentes R, G e B de um pixel.
- `indicator` dado um personagem (*e.g.* 1,2,3) e um pixel com componentes R, G, B, indica se esse pixel pertence ou não à personagem.
- `location` calcula o "centro de massa" para um certo personagem p. Note que, como as funções só podem retornar um valor, as coordenadas do "centro de massa" têm de ser passadas por referência para serem preenchidas pela função.

As funções podem assumir internamente um tamanho de imagem fixo, ou receber o tamanho nos parâmetros. A imagem fornecida tem dimensão 320×180.

Antes de começar a programar em Assembly, sugere-se que primeiro implemente o programa na linguagem C.

Comente o código como no seguinte exemplo:

```
#########################################################
# Funcao: media
# Descricao: Esta funcao calcula a media de 2 numeros.
# Argumentos:
#   a0 - primeiro numero
#   a1 - segundo numero
# Retorna:
#   a0 - media
#########################################################
media:
        add a0, a0, a1
        srai a0, a0, 1
        ret
```

Escreva o relatório em LaTeX. Entregue apenas o PDF final.

10 Submissão do trabalho

- O trabalho deverá ser realizado em **grupos de 2 alunos**.
- O trabalho deve ser submetido no moodle num único ficheiro comprimido (rar, zip, tgz, etc) com nome formado pelos números dos alunos por ordem, por exemplo `11111-22222.zip`. Apenas um dos alunos do grupo deve submeter.
- O trabalho é longo. Devem começar já!
- Qualquer situação de fraude implica a reprovação à unidade curricular e será reportada para instauração de procedimento disciplinar, conforme estipulado no regulamento académico.

Bom trabalho! 👍
Miguel Barão