Uma linguagem de programação para processamento de imagens

Henrique Miyamoto e Thiago Benites

I. Contextualização

Apresentamos uma linguagem de programação voltada para o processamento de imagens digitais. Ela foi implementada com uma gramática livre de contexto, usando analisadores léxicos e sintáticos. Como uma linguagem de propósito específico, pretende-se que seja intuitiva para o usuário [1]. As funcionalidades que nossa linguagem é capaz de executar, assim como as respectivas sintaxes são apresentadas na Tabela I.

TABLE I
Funcionalidades e sintaxe da linguagem de programação

Funcionalidade	Sintaxe
Salvar uma imagem	destino.jpg = origem.jpg
Alterar brilho	destino.jpg = origem.jpg * float destino.jpg = origem.jpg / float
Detectar valor máximo	[origem.jpg]

Nos processos de salvar uma imagem e de alterar seu brilho, são feitas cópias do arquivo original. Para detecção do valor máximo da imagem, calculamos o valor de cada pixel como a intensidade na conversão do espaço RGB para o HSI $I = \frac{1}{3}(R+G+B)$ [2].

II. Demonstração

Para demonstração das funcionalidades, tomemos por base a imagem em sua forma original apresentada na Figura 1(a). Quando aplicado brilho de fator multiplicativo 2, tem-se o resultado apresentado na Figura 1(b). Já ao dividir pelo mesmo fator, obtém-se o resultado da Figura 1(c).

(a) (b) (c)

Fig. 1. Imagens: (a) original, (b) com brilho dobrado e (c) reduzido à metade.

Para as três figuras é possível aplicar a função que detecta o máximo valor de brilho, de acordo com os resultados a seguir.

```
./main
[figura1.jpg]
Intensidade maxima: 254.00
[figura2.jpg]
Intensidade maxima: 255.00
[figura3.jpg]
Intensidade maxima: 133.00
```

III. Análise

Comparamos a ideia de usar um comando específico para alterar o brilho com aplicações equivalentes de propósito geral, como OpenCV. Em sua versão para C++, a aplicação de brilho a uma imagem pode ser feita através do comando convertTo(), com a seguinte sintaxe [3]:

```
void Mat::convertTo(OutputArray m, int rtype,
double alpha=1, double beta=0 ) const
```

em que alpha e beta são fatores de escala que realizam a operação $g(x) = \alpha f(x) + \beta$, sobre a imagem $f(x)$, resultando na imagem $g(x)$ [4].

O comando utilizado na nossa linguagem para tal função é mais simples, no sentido de que tem menos argumentos e lida com menos conceitos de programação que o do OpenCV, exigindo menos conhecimento do usuário. Por exemplo, no comando do OpenCV, é necessário entender que aplica-se uma transformação linear à imagem, com parâmetros que definem contraste e brilho, além de lidar com a imagem na forma matricial. Na nossa sintaxe, é necessário utilizar apenas os nomes dos arquivos e a intensidade de brilho que se deseja aplicar, ficando ocultos para o usuário final o formato como a imagem é tratada e os procedimentos realizados.

Se por um lado comandos como o do OpenCV podem ser aplicados de forma mais geral, sintaxes mais diretas oferecem ganho em expressividade e facilidade no uso. Isso aumenta a quantidade de usuários em potencial da linguagem e permite uma maior produtividade, conforme afirma [1].

* Os códigos do projeto estão disponíveis em `https://github.com/miyamotohk/linguagem-processamento-imagem`.

Referências

[1] MERNIK, M., HEERIN, J., SLOANE, A. M. When and how to develop domain-specific languages. In: *ACM Computing Surveys (CSUR)*. Nova York, vol. 37, ed. 4, p. 316-344, dez. 2005.

[2] Wikipedia. *RGB color model*. Disponível em: `https://en.wikipedia.org/wiki/RGB_color_model`. Acesso em: 10 set. 2017.

[3] OpenCV. *Basic Structures*. Disponível em: `http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html`. Acesso em: 11 set. 2017.

[4] OpenCV. *Changing the contrast and brightness of an image!*. Disponível em: `http://docs.opencv.org/2.4/doc/tutorials/core/basic_linear_transform/basic_linear_transform.html`. Acesso em: 11 set. 2017.