# Projeto Demonstrativo 1 - Introdução ao OpenCV

Samuel Venzi Lima Monteiro de Oliveira
14/0162241
samuel.venzi@me.com

29-03-2016

## 1 Objetivos

O objetivo deste projeto é usar dos conhecimentos de OpenCV para criar imagens manipulando matrizes e ainda analisar as diferenças nos códigos e execuções para as versões 1.x e 2.x do OpenCV.

## 2 Introdução

Em visão computacional a manipulação de imagens é imprescindível. Com esse tipo de análise e processo, várias aplicações se tornam viáveis. Diminuição de ruídos, melhoramento de cor e criação de desenhos em softwares são alguns exemplos onde tal manipulação é essencial. Dessa forma, é necessário conhecer e entender o processo de análise de imagens, sejam prontas ou criadas.

No computador, imagens são vistas como matrizes onde cada pixel é representado por uma coordenada na matriz e o valor dessa coordenada passa alguma informação em relação àquele pixel, como cor e intensidade. Dessa forma, ao ler ou criar um arquivo de imagem, é essencial buscar essas informações para que possam ser traduzidas em luz pela tela do computador. A plataforma OpenCV permite com facilidade esse tipo de manipulação.

### 2.1 Criação de matrizes de imagem

A estrutura básica da uma imagem gerada por computador é uma matriz. Portanto, é razoável pensar em cada pixel como um elemento matricial. Para se entender melhor o funcionamento desse sistema, definida uma matriz de ordem 5, por exemplo, pode-se dizer que cada um dos 25 elementos presentes poderá assumir valor 0 ou valor 1, onde zero é pixel apagado e 1 é pixel acesso. Logo, se a matriz tiver a seguinte configuração:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

a imagem formada será uma cruz definida pelos elementos que possuem valor 1, isto é, os respectivos pixels se acenderão.

Estendendo esse método para a criação de imagens mais complexas, cada elemento pode assumir um espectro de valores maior que (0,1). Cada valor desse novo espectro pode, por exemplo, representar uma intensidade de luminosidade do pixel e gerar uma imagem preto e branco. É justamente esse método que será utilizado neste experimento de criação de imagens.

### 2.2 OpenCV

OpenCV é uma plataforma livre que trabalha com algoritmos de visão computacional, portanto, é a ferramenta adequada para a execução deste experimento. Essa plataforma, originalmente em sua versão 1, utilizava a linguagem C juntamente com suas bibliotecas, porém, com sua evolução, na versão 2 passou ter suporte para a linguegem C++, o que tornou os códigos mais concisos e mais eficientes.

Neste experimento, com um código escrito para a versão 1, em C, será feita a tradução para a linguagem C++ da versão 2 e a comparação da eficiência de ambos as versões do mesmo algoritmo.

## 3 Materiais e Metodologia

### 3.1 Materiais

- Computador com ambiente Linux (Ubuntu)
- OpenCV

### 3.2 Metodologia

Primeiramente foi feita a análise do código da versão 1 para completo entendimento do funcionamento do algoritmo. Com isso feito, foi possível, com o uso de novas funções, escrever o código da versão 2 com a linguagem C++. Após isso, foi feita a comparação no tempo de execução de cada programa para determinar sua eficiência.

## 4 Resultados

### 4.1 Versão 1.x

```
[language=C]
#include <cv.h>
#include <highgui.h>
#include <stdio.h>

double gettime();

int main()
{
   IplImage *cvImg; // Objeto usado para armazenar a imagem
   CvSize imgSize;
   int i1 = 0, j1 = 0;
   double t;
   imgSize.width = 640;
   imgSize.height = 480;
   double start = gettime();
   cvImg = cvCreateImage( imgSize, 8, 1 );
   for ( i1 = 0; i1 < imgSize.width; i1++ )
     for ( j1 = 0; j1 < imgSize.height; j1++ )
   ((uchar*)(cvImg->imageData + cvImg->widthStep*j1))[i1] = (char)((i1 * j1)%256);
   cvNamedWindow( "Abrindo a Imagem Gerada...", 1 );
   cvShowImage( "Abrindo a Imagem Gerada...", cvImg );
   double stop = gettime();
   t = stop - start;
   cvWaitKey(4000);
```

```
   cvDestroyWindow( "image" );
   cvReleaseImage( &cvImg );
   printf("Tempo de execução: %f\n", t);
   return 0;
}

double gettime(){
struct timespec t;

if(clock_gettime(CLOCK_REALTIME, &t) != 0)
return 0;
return (t.tv_sec + (t.tv_nsec/1000000000.0));
}
```

O código acima, escrito em C, cria uma matriz de imagem 640 por 480 pixels e atribui um valor específico a cada um de seus elementos usando a seguinte função: dado um elemento na linha i e coluna j, o valor deste elemento é dado pelo resto da divisão de $(i * j)$ por 256. Portanto, um elemento pode assumir valores de 0 a 255. Neste caso, esses valores representam a intensidade do pixel no espectro preto e branco.

Primeiramente, o programa cria uma estrutura que armazenará a imagem (de acordo com seu tamanho) e passar por um loop em que o valor de cada pixel é definido um a um. Com a imagem gerada, o programa mostra a imagem na tela e a mantém aberta por 4 segundos. Para determinar o tempo de execução e padronizar a maneira de medição, o programa mede o tempo de criação da matriz e preenchimento da mesma, por ser a parte computacionalmente pesada. Para isso, uma função que recupera o instante atual foi escrita. Fazendo isso uma vez antes da criação e outra após o preenchimento, basta subtrair os valores para achar o tempo de geração.

Table 1: Tempo de geração(em segundo)

| execução | 640 x 480 | 1280 x 720 | 4320 x 1280 |
|---|---|---|---|
| 1 | 0.031544 | 0.062557 | 0.147598 |
| 2 | 0.036091 | 0.052864 | 0.136265 |
| 3 | 0.033601 | 0.044304 | 0.150777 |
| 4 | 0.041316 | 0.053736 | 0.160500 |
| 5 | 0.037135 | 0.044105 | 0.184669 |
| 6 | 0.036870 | 0.051061 | 0.177290 |
| 7 | 0.038892 | 0.051338 | 0.160479 |
| 8 | 0.027884 | 0.054060 | 0.163546 |
| 9 | 0.036561 | 0.053173 | 0.152138 |
| 10 | 0.036696 | 0.064611 | 0.183972 |

Como se pode perceber, quanto maior o tamanho da imagem a ser gerada, maior o tempo para preenche-la. Também é possível observar que os valores para cada um dos três tamanhos flutuam em torno de um valor, para saber que valor é esse basta fazer uma análise estatística simples.

Para o tamanho 640 x 480, o computador utilizado leva em média 0.035659 segundos para preencher a matriz. O desvio padrão da média é 0.002790 para este conjunto de medidas. Com a imagem um pouco maior, 1280 x 720, a média do tempo é 0.053181 segundos com desvio padrão médio de 0.004448. Finalmente, para o meio tamanho de imagem, 4320x1280, a média é 0.161723 e o desvio padrão da média é 0.012517.

## 4.2 Versão 2.x

```
[language=C++]
```

```
#include <stdio.h>
#include <opencv2/opencv.hpp>
#include <time.h>

double gettime();

int main(int argc, char const *argv[])
{
int i, j;
double start, stop, t;
int width = 640, height = 480;
start = gettime();
cv::Mat img(height, width, CV_8U);
for (i = 0; i < height; ++i)
{
for (j = 0; j < width; ++j)
{
img.at<uchar>(i,j) = (char)(i*j)%256;
}
}
stop = gettime();
imshow("Imagem gerada", img);
t = stop - start;
cv::waitKey(0);
printf("Tempo de execução: %f segundos\n",t);
return 0;
}

double gettime(){
struct timespec t;
if(clock_gettime(CLOCK_REALTIME, &t) != 0)
return 0;
return (t.tv_sec + (t.tv_nsec/1000000000.0));
}
```

A primeira vista podemos citar diferenças no código. Desta vez, não é necessário o uso de ponteiros para criar e acessar a matriz. Usando a estrutura matricial Mat com os parâmetros necessários, a matriz é criada. O algoritmo é essencialmente o mesmo, porém otimizado como será visto na tabela de valores.

Table 2: Tempo de geração(em segundo)

| execução | 640 x 480 | 1280 x 720 | 4320 x 1280 |
|---|---|---|---|
| 1 | 0.005548 | 0.016425 | 0.083487 |
| 2 | 0.007692 | 0.014425 | 0.084515 |
| 3 | 0.004866 | 0.016808 | 0.074352 |
| 4 | 0.004943 | 0.015745 | 0.070159 |
| 5 | 0.007812 | 0.013459 | 0.085039 |
| 6 | 0.005723 | 0.016592 | 0.084341 |
| 7 | 0.005460 | 0.015385 | 0.080928 |
| 8 | 0.007498 | 0.016842 | 0.084687 |
| 9 | 0.005318 | 0.014643 | 0.083418 |
| 10 | 0.005198 | 0.015243 | 0.080235 |

Pode-se perceber uma grande diferença entre os valores apresentados na tabela 1 para os respectivos tamanhos. Acontece também algo já esperado, o aumento do tempo com o tamanho da imagem.

Fazendo a mesma análise para a versão 2.x, o tamanho 640 x 480 tem média de tempo de 0.006006 segundos e desvio padrão da média de 0.000997. Para o tamanho 1280 x 720, a média foi 0.015557 e o desvio padrão médio de 0.000926. E, para o tamanho 4320 x 1280, a média foi 0.081116 e o desvio padrão médio de 0.003758.

# 5 Discussão e Conclusões

Recuperando os valores médios de execução calculados na seção anterior, observa-se o comportamento esperado. Os valores de tempo para para os respectivos tamanhos são menores para a versão otimizada 2.x do OpenCV.

Table 3: Tempos médios de execução (segundos)

| tamanho | 1.x | 2.x |
|---|---|---|
| 640 x 480 | 0.035659±0.002790 | 0.006006±0.000997 |
| 1280 x 720 | 0.053181±0.004448 | 0.015557±0.000626 |
| 4320 x 1280 | 0.161723±0.012517 | 0.081116±0.003758 |

Claramente, a versão 2.x permite um código mais eficiente e computacionalmente mais leve que a versão 1.x. Com o aumento do tamanho da imagem o tempo de execução também aumenta, já que exige mais recursos do computador. Desta forma, a versão 2.x é a melhor opção de uso em visão computacional.

# 6 Bibliografia

A bibliografia principal utilizada foram sites da internet.

http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_gcc_cmake/linux_gcc_cmake.html#linux-gcc-usage

http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html

http://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html