SISTEMAS SENSORIAIS

AULA 1 - *Introdução do Processamento de Imagem*

Objectivo:

Pretende-se que na primeira parte desta aula haja uma familiarização com a ferramenta de desenvolvimento, o Microsoft Visual Studio C# e numa segunda parte seja explorada a biblioteca de processamento de imagem EmguCV (versão C# da biblioteca OpenCV (Open Computer Vision)) e enriquecida a aplicação de processamento de imagem disponibilizada.

Procedimento:

1 - Para conhecer a ferramenta de desenvolvimento, Microsoft Visual Studio C#, recomenda-se que instale a aplicação exemplo disponibilizada e siga os seguintes passos:

1. Explore as várias formas de aceder às classes, janelas, menus, etc. O funcionamento da aplicação fornecida é baseado em eventos gerados por menus, botões, etc. Para aceder aos métodos que tratam esses eventos prima com o rato sobre o menu ou botão correspondente e será encaminhado para a função que trata esse evento.
2. Visualize também a janela contendo as propriedades dos componentes (View -> *Properties Window*).
   - Experimente visualizar as propriedades do componente *ImageViewer* que se encontra no interior na janela. Este componente é utilizado para a visualização de imagens (BMP, JPG ou de outro tipo), sendo por isso fundamental para o processamento de imagem.

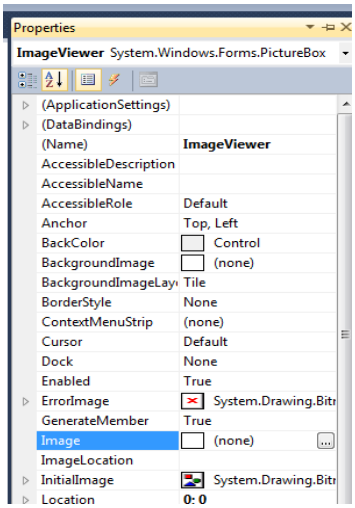

Figura 1

2 - Para testar a aplicação fornecida experimentem executar algumas das funcionalidades básicas do processamento de imagem disponibilizadas e comparem o tempo de execução das funções 2 e 3:

1. Abrir imagem
2. Converter para cinza
3. Negativo da imagem
4. *Undo* (repor a última imagem)

Manipulação de imagem:

Existem pelo menos duas formas de aceder ao conteúdo dos pixéis:

- por primitivas da biblioteca EmguCV
  - `img[ y , x ] = new Bgr(127,127,127);`
- por acesso directo à memória
  - `dataPtr[0] = 127;`
  - `dataPtr[1] = 127;`
  - `dataPtr[2] = 127;`

O acesso direto à memória, apesar de ser um pouco mais complexo, é muito mais rápido e eficiente o que resulta em termos de tempo de execução menores e será portanto o método escolhido para ser usado nas aulas.

O espaço de memória reservado para armazenar a imagem armazena cada pixel com as suas correspondentes componentes de cor RGB por ordem inversa: BGR (Blue, Green e Red), sendo 1 byte (0..255) para cada componente. O pixel seguinte virá imediatamente a seguir e após passar todos os pixéis de uma linha, aparecem os pixéis da linha seguinte.

Contudo, por razões de optimização da informação em memória, as linhas têm de ser múltiplas de 4 ou 8 bytes para estarem alinhadas e serem de rápido acesso.

Assim, no final de uma linha de pixéis ainda podem aparecer alguns bytes que servem apenas para garantir o alinhamento (*padding*), ou seja, se a largura da imagem multiplicada pelo número de canais (RGB) não for múltipla de 4 ou 8, serão acrescentados bytes até perfazer esse alinhamento (figura 2).

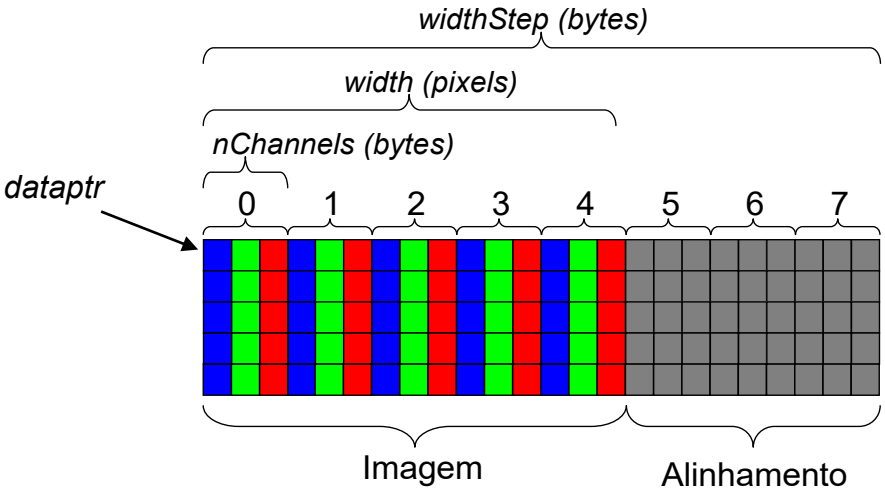

Figura 2

O primeiro passo para manipular o conteúdo dos pixéis consiste em obter o endereço (`dataptr`) do primeiro pixel (topo superior esquerdo) e depois avançar com esse apontador para aceder aos restantes pixéis.

```
MIplImage m = img.MIplImage;
byte* dataPtr = (byte*)m.imageData.ToPointer(); // obter apontador
```

A estrutura de dados `MIplImage` contém diversos parâmetros que descrevem como está organizada a imagem em memória, destacando-se os seguintes campos:

- `width` - indica a largura da imagem (em pixéis);
- `widthStep` - indica qual a largura total de uma linha (em bytes);
- `nChannels` - que indica o número de canais de cor, (RGB = 3).

Assim, quando ao trabalhar com a estrutura `dataptr` for necessário avançar ou recuar uma linha, é preciso usar o campo `widthStep`.

- Avançar uma linha de pixéis completa:

```
dataPtr += m.widthStep;
```

- Avançar o número de pixéis de alinhamento para passar à linha seguinte:

```
int padding = m.widthStep - m.nChannels * m.width;
dataPtr += padding;
```

3 – Para aprender a utilizar este modo de manipulação de pixéis crie uma nova versão da função negativo da imagem, mas que aceda diretamente à memória.

4 – Implemente uma função que mostre apenas uma das componentes de cor da imagem. Para tal, copie o conteúdo da componente selecionada para as restantes.

Nota: o objetivo é obter uma imagem em tons de cinzento com a informação de uma das componentes de cor (Red, Green ou Blue)

5 – Implemente uma função que ajuste o brilho e o contraste da imagem com os valores inseridos pelo utilizador. Para alterar o valor do pixel use a expressão:

pixel_modificado(x,y) = contraste * pixel(x,y) + brilho

Nota 1: o valor de brilho é inteiro, pode ser positivo ou negativo e numa gama entre 0 e 255.

Nota 2: o valor de contraste é do tipo *float*, positivo e numa gama entre 0 e 3.

Nota 3: o valor do pixel modificado é limitado ao intervalo [0 , 255], devendo saturar em zero e 255.

Nota 4: Use a janela InputBox para pedir valores ao utilizador.

Exemplo:

```
InputBox form = new InputBox("brilho?");
form.ShowDialog();

int brilho = Convert.ToInt32(form.ValueTextBox.Text);
```