

# Universidade Federal do Rio Grande do Sul

## INF01046 - Fundamentos de Processamento de

### Imagens - Turma B - Trabalho 1

Gabriela Copetti Maccagnan

20/10/2024

## 1 Operações Básicas sobre Imagens Digitais

Este trabalho tem como objetivo implementar operações básicas sobre imagens digitais, utilizando a linguagem C++, juntamente com as bibliotecas *OpenCV* para auxílio nas operações de processamento de imagens e a biblioteca *cvui* (<https://fernandobevilacqua.com/cvui/>) para a construção de uma interface para o programa. A escolha pela biblioteca para a construção da interface se deu por ela ser header-only, sem dependências externas, além da simplicidade de implementação e utilização.

O programa foi construído utilizando CMake, portanto possui um arquivo *CMakeLists.txt* e pode ser rodado a partir dos comandos na pasta root:

- cmake -B build (para fazer o build do programa)
- cmake --build build --config Release (para criar o executável na pasta \build\Release)
- .\build\Release\FPI\_T1.exe (para rodar o executável)

O programa conta com um arquivo main.cpp, onde foi desenhada a interface e de onde são chamadas as funções de processamento a partir do clique nos botões da tela, um arquivo imageProcessing.cpp, onde foram definidas as funções relacionadas às operações de processamento das imagens como espelhamentos e aplicações de filtros e, por fim, um arquivo filesOperations.cpp, usado para realizar operações como abrir e salvar arquivos no computador. Para fechar o programa, basta clicar na tecla *esc*.

### 1.1 Parte I – Leitura e Gravação de Arquivos de Imagens

Existem duas opções de carregamento de imagens no programa: carregamento de uma imagem qualquer do computador ou carregamento de uma imagem padrão disponibilizada para os testes do trabalho. As imagens teste podem ser visualizadas na Figura 1.



Figure 1: Imagens teste

Para a leitura de uma imagem qualquer basta o usuário clicar no botão "Open Image" e selecionar a imagem desejada e para abrir uma das imagens de teste o usuário pode clicar em um dos três botões que se encontram na seção "Use Default Images", sendo que o nome de cada botão se refere à imagem que será aberta.

O usuário pode modificar e trocar a imagem selecionada quantas vezes quiser. Para trocar a imagem basta clicar para abrir uma nova, substituindo a original. Para modificar ou salvar uma imagem o usuário precisa primeiro criar uma cópia da mesma clicando no botão "Copy Image". A cópia da imagem será visualizada em uma janela separada da imagem original (Figura 13) e qualquer operação escolhida pelo usuário irá modificar a cópia da imagem, nunca a original. Portanto, se o usuário deseja resetar a imagem ao seu estado original para realizar novas operações sobre ela, basta clicar em "Copy Image" novamente.

As imagens geradas no programa podem ser salvas clicando no botão "Save Image" e podem assumir qualquer formato (.jpeg, .png ou .jpg). O usuário pode escolher o nome desejado para salvar a imagem criada, nesse caso devendo também informar no campo do nome a extensão da imagem. Caso o usuário não queira informar um nome, a imagem será salva com um nome padrão e a extensão será .jpeg.

```

void showImage(const Mat& img, const string& title = "Copied Image") {
    if (!img.empty()) {
        imshow(title, img);
    } else {
        cerr << "Error: Cannot open image" << endl;
    }
}

void readImage(Mat &img, const string& imagePath) {
    img = imread(imagePath);
    showImage(img, "Original Image");
}

void copyImage(Mat &original, Mat &copy) {
    copy = original.clone();
}

void saveImage(string filePath, Mat& img) {
    imwrite(filePath, img);
    cout << "Image saved at: " << filePath << endl;
}

```

Figure 2: Definição das funções de apresentação, leitura, cópia e salvamento de uma imagem

Após salvar as imagens com a extensão .jpeg é possível notar uma diferença no seu tamanho, o que se deve a natureza do formato escolhido. Esse é um formato de compressão com perda, o que seignifica que ocorre uma redução de dados durante a compressão e, portanto, salvar imagens como .jpeg acaba modificando características como qualidade e tamanho do arquivo.





 Space_187k		12/10/2024 19:25	Arquivo JPG	187 KB
 spaceCopyJPEG		20/10/2024 20:09	Arquivo JPEG	107 KB
 spaceCopyPNG		20/10/2024 15:10	Arquivo PNG	381 KB

Figure 3: Comparação dos tamanhos finais de imagens salvas com diferentes formatos

## 1.2 Parte II – Leitura, Exibição e Operações sobre Imagens

### 1.2.1 Espelhamento Horizontal e Vertical

Para realizar o espelhamento da imagem escolhida, o usuário precisa primeiro copiar a imagem conforme já foi descrito e, em seguida, clicar nos botões "Vertical" ou "Horizontal" que se encontram na seção chamada "Mirror" (Figura 12).

```

Mat mirrorVertical(const Mat& img) {
    Mat mirroredImage(img.size(), img.type());

    for (int y = 0; y < img.rows; y++) {
        const uchar* oldRow = img.ptr<uchar>(y);
        uchar* newRow = mirroredImage.ptr<uchar>(img.rows - 1 - y);

        memcpy(newRow, oldRow, img.cols * img.channels());
    }

    return mirroredImage;
}

Mat mirrorHorizontal(const Mat& img) {
    Mat mirroredImage(img.size(), img.type());

    for (int y = 0; y < img.rows; y++) {
        const uchar* oldRow = img.ptr<uchar>(y);
        uchar* newRow = mirroredImage.ptr<uchar>(y);

        for (int x = 0; x < img.cols; x++) {
            memcpy(&newRow[(img.cols - 1 - x) * img.channels()], &oldRow[x * img.channels()], img.channels());
        }
    }

    return mirroredImage;
}

```

Figure 4: Definição das funções de espelhamento



Figure 5: Comparação da imagem original com a resultante do espelhamento vertical





Figure 6: Comparação da imagem original com a resultante do espelhamento horizontal

O espelhamento pode ser combinado, clicando nos dois botões.

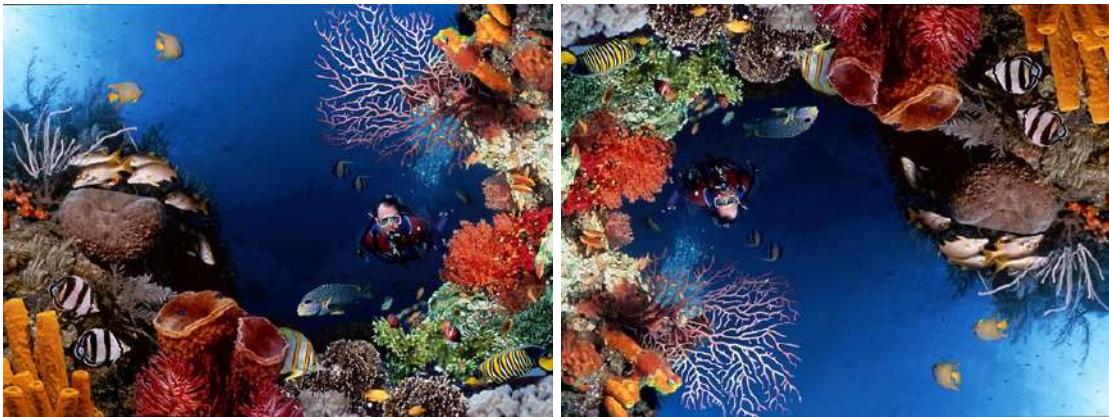


Figure 7: Comparação da imagem original com a resultante do espelhamento combinado, ou seja, tanto horizontal como vertical

### 1.2.2 Conversão para Tons de Cinza

Para modificação de uma imagem colorida para uma imagem em tons de cinza foi usada a equação  $L = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$  que representa os pesos aplicados a cada um dos três canais de cores da imagem: R (red), G (green) e B (blue).

```

Mat convertToGS(const Mat& img) {
    Mat grayImage(img.size(), CV_8UC1);

    for (int y = 0; y < img.rows; y++) {
        for (int x = 0; x < img.cols; x++) {
            Vec3b pixel = img.at<Vec3b>(y, x);
            uchar R = pixel[2];
            uchar G = pixel[1];
            uchar B = pixel[0];

            uchar L = 0.299 * R + 0.587 * G + 0.114 * B;

            grayImage.at<uchar>(y, x) = L;
        }
    }

    return grayImage;
}

```

Figure 8: Definição da função de conversão para tons de cinza

Na interface, o usuário pode aplicar o filtro de escala de cinza em uma imagem realizando os seguintes passos: primeiro escolher uma imagem; depois copiar a imagem; por fim, clicar no botão "Grayscale" dentro da seção "Filters".



Figure 9: Comparação da imagem original com a resultante da aplicação do filtro grayscale

A aplicação de um filtro de escala de cinza em uma imagem que já se encontra nessa escala não irá surtir novas alterações, pois todos os pixels da imagem em tons de cinza já apresentam um mesmo valor variando de 0 a 255 em cada um dos três canais de cor. Portanto, ao aplicar a equação acima sobre um pixel dessa imagem teremos que  $R = G = B = L$ , o que não altera o valor do pixel.

### 1.2.3 Quantização sobre as Imagens

Para a quantização de uma imagem em escala de cinza, foram definidos algumas variáveis para: os valores de  $t1$  e  $t2$ , ou seja, os tons com o menor e o maior valor de intensidade na imagem;  $tam\_int = (t2 - t1 + 1)$ , a quantidade de tons presentes na imagem;  $n$ , o número de tons a ser utilizado no processo de quantização conforme definido pelo usuário; e o novo valor de cada pixel, através do valor inteiro mais próximo do centro do intervalo do bin que contém o valor do pixel original.

```
Mat quantization(const Mat& img, const int& n) {
    Mat quantizedImage(img.size(), img.type());

    uchar t1 = 255;
    uchar t2 = 0;

    for (int y = 0; y < img.rows; y++) {
        for (int x = 0; x < img.cols; x++) {
            uchar pixel = img.at<uchar>(y, x);
            if (pixel < t1) {
                t1 = pixel;
            }
            if (pixel > t2) {
                t2 = pixel;
            }
        }
    }

    int tam_int = t2 - t1 + 1;

    if (n >= tam_int) {
        return img;
    }

    float tb = static_cast<float>(tam_int) / n;

    for (int y = 0; y < img.rows; y++) {
        for (int x = 0; x < img.cols; x++) {
            uchar pixel = img.at<uchar>(y, x);

            int binIndex = static_cast<int>((pixel - t1 + 0.5) / tb);

            int quantizedPixel = static_cast<int>(t1 + (binIndex * tb) + (tb / 2));

            quantizedPixel = min(max(quantizedPixel, 0), 255);

            quantizedImage.at<uchar>(y, x) = static_cast<uchar>(quantizedPixel);
        }
    }

    return quantizedImage;
}
```

Figure 10: Definição da função de quantização

Na interface, o usuário pode realizar a ação de quantização escolhendo uma

imagem, copiando a imagem, aplicando sobre ela o filtro de escala de cinza, definindo o número de tons desejados no componente *counter* da interface e, uma vez definido um valor acima de 0, clicar no botão "Quantization".

As imagens resultantes da aplicação de três valores diferentes de tons na quantização podem ser visualizadas abaixo.



Figure 11: Imagens resultantes da quantização de uma das imagens teste para  $n = 5$ ,  $n = 15$  e  $n = 30$ , respectivamente

#### 1.2.4 Salvamento da Imagem Resultante

A opção de salvar a imagem resultante das operações de processamento já foi brevemente descrita na primeira seção, podendo o usuário a qualquer momento clicar em "Save Image" e definir o nome e local de salvamento da imagem gerada, dado que ela tenha sido copiada primeiramente.

### 1.3 Interface

Abaixo se encontram algumas imagens da interface criada para o programa.



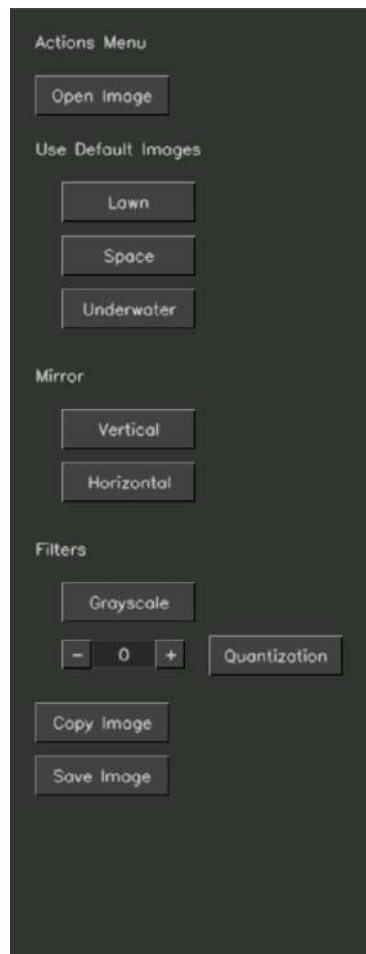


Figure 12: Menu de ações disponibilizadas para o usuário



Figure 13: Abertura da imagem original e da modificada pelo usuário na interface

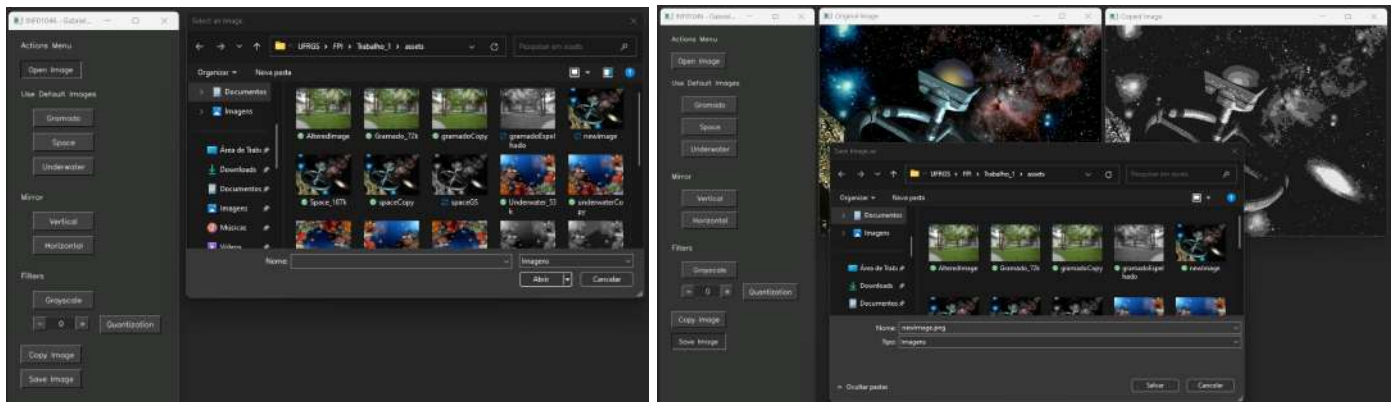


Figure 14: Captura do momento de abertura de uma imagem qualquer no computador e de salvamento da imagem pós alterações, respectivamente

## 1.4 Dificuldades encontradas

A maior dificuldade encontrada durante o trabalho se deve a pouca familiaridade com a linguagem C++ e suas bibliotecas de construção de interfaces. Infelizmente por questões de tempo, outras tentativas de uso de bibliotecas mais complexas que permitem a criação de interfaces mais atraentes precisaram ser postas de lado. Acredito que para os próximos trabalhos, além de ser necessário um melhor gerenciamento de tempo, a linguagem já será algo mais natural de se trabalhar e deixará de ser um problema.