UFV - Universidade Federal de Viçosa
DPI - Departamento de Informática
Discente - Pedro Santos Teixeira
Matrícula: 116224

INF 110 - Programação I
Trabalho 3

Agosto de 2024

Relatório de Desenvolvimento do Programa de Processamento de Imagens PNM

.

Lista de figuras

Sumário

1. Introdução

Este documento apresenta o desenvolvimento de um programa destinado ao processamento de imagens no formato PNM (Portable Anymap). O programa foi criado como parte do trabalho prático da disciplina de Programação 1, da Universidade Federal de Viçosa (UFV). Seu objetivo é aplicar diversas operações de processamento de imagem, como clareamento, escurecimento, espelhamento, ajuste de contraste, filtro de Prewitt, conversão para tons de cinza e aplicação de efeito negativo.

A implementação visa exercitar conceitos fundamentais de programação, incluindo leitura e escrita de arquivos, manipulação de matrizes, uso de estruturas de dados personalizadas e implementação de algoritmos de processamento digital de imagens. Esse trabalho foi elaborado como item avaliativo da matriz curricular do curso de Ciência da Computação da UFV.

1.1 Ambiente de desenvolvimento

Para a elaboração do código o autor deste trabalho optou pela utilização da IDE (Integrated Development Environment) Dev-C++ que utiliza o compilador TDM-GCC 4.9.2 64-bit Release, realizado e testado no sistema operacional Windows 10 Home Single Language versão 22H2.

2. Descrição do programa

O programa é baseado em um menu interativo que permite ao usuário escolher diferentes operações para aplicar em imagens no formato PNM. A imagem original é lida, processada de acordo com a escolha do usuário, e um novo arquivo com a imagem modificada é gerado. Para tal, é de notória importância salientar que para o funcionamento ocorrer de forma adequada o usuário deve inserir o nome do arquivo de entrada bem como o nome do arquivo de saída.

```
//escopo de constantes
const int MAX = 500;
const int MAXALTURA = 500;
const int MAXLARGURA = 500;
const int MAX_PIXEL = 255;

//escopo de funcoes
void clearScreen(); //funcao auxiliar de limpar tela em win, ubuntu, mac e linux
void arqRead(const char* arqName, string& tipo, int& largura, int& altura, int& valor,
    Cor img[MAXALTURA][MAXLARGURA]); //funcao para abertura e leitura da matriz da imagem
void arqRecord(const char* arqNew, string& tipo, int& largura, int& altura,
    Cor img[MAXALTURA][MAXLARGURA]);//funcao para leitura da nova imagem e fechamento

int main(){
    //escopo de variaveis
    char arqName[MAX]; //nome do arquivo original
    char arqNew[MAX];  //nome do arquivo que será criado
    short choice;     //escolha do menu
    string tipo;     //string tipo de imagem(P2 ou P3)
    char comment[200], c; //variaveis auxiliares
    int largura, altura;
    int valor, fator;
    Cor img[MAXALTURA][MAXLARGURA]; //declaracao da imagem atraves do tipo Cor (tanto cinza quanto colorida)

    //variaveis de fluxo stream
    ifstream arqIn;
    ofstream arqOut;
```

Figura 1: Escopo de declarações utilizadas para a implementação do código.

A estrutura modular do programa facilita a execução de múltiplas operações na mesma imagem, com a possibilidade de alternar entre diferentes opções até o encerramento da execução. A implementação de uma boa estrutura de visualização foi essencial para a elaboração deste trabalho, visto que um bom entendimento do usuário final é fundamental para o funcionamento adequado das ferramentas.

```
do{
    inicio:
    system("color 4F"); //troca a cor da tela de saida no windows;
    do {
        clearScreen();
        cout << "|Bem-vindo(a) aos nossos servicos! Favor inserir o valor desejado:\n\n";

        cout << "|-----------> |MENU| <-----------|\n";
        cout << "|01|-> Clareamento.\n";
        cout << "|02|-> Escurecimento.\n";
        cout << "|03|-> Negativo.\n";
        cout << "|04|-> Espelhar.\n";
        cout << "|05|-> Filtro de Prewitt.\n";
        cout << "|06|-> Contraste.\n";
        cout << "|07|-> Tons de cinza.\n";
        cout << "|08|-> Sair.\n";
        cout << "|--------------------------------|\n";
        cout << "\n|Sua escolha: ";
        cin >> choice;

        if(choice < 1 || choice > 8){
            cout << "|Escolha invalida. Tente novamente.\n\n";
            system("pause");
        }

    } while (choice < 1 || choice > 8);

    if(choice!=8){
        cout << "|Insira o nome do arquivo: ";
        cin >> arqName;
        cout << "|Insira o nome do novo arquivo: ";
        cin >> arqNew;
    }
    clearScreen();
```

Figura 2: Implementação do menu de opções.

```
|Bem-vindo(a) aos nossos servicos! Favor inserir o valor desejado:

|-----------> |MENU| <-----------|
|01|-> Clareamento.
|02|-> Escurecimento.
|03|-> Negativo.
|04|-> Espelhar.
|05|-> Filtro de Prewitt.
|06|-> Contraste.
|07|-> Tons de cinza.
|08|-> Sair.
|--------------------------------|

|Sua escolha:
```

Figura 3: Visualização do usuário final da tela de menu.

2.1 Funcionalidades implementadas

2.1.1 Clareamento

- Aumenta o brilho da imagem. O usuário insere um fator de brilho(1-100), e o programa ajusta os valores dos pixels, incrementando-os proporcionalmente até o valor máximo permitido (255).

2.1.2 Escurecimento

- Diminui o brilho da imagem. Um fator de escurecimento é fornecido pelo usuário(1-100), e os valores dos pixels são reduzidos proporcionalmente, respeitando o valor mínimo (0).

2.1.3 Negativo

- Inverte as cores da imagem, criando um efeito negativo. Cada valor de pixel é subtraído de 255, resultando em uma inversão total dos tons.

2.1.4 Espelhamento horizontal

- Reflete a imagem horizontalmente, invertendo a posição dos pixels em cada linha.

2.1.5 Filtro de Prewitt

- Aplica o filtro de Prewitt para detecção de bordas, utilizando matrizes convolucionais (`Gx` e `Gy`) que calculam os gradientes horizontais e verticais da imagem.

2.1.6 Ajuste de contraste

- Ajusta o contraste da imagem com base em um fator inserido pelo usuário. O contraste é modificado com base na diferença entre os valores dos pixels e o brilho médio da imagem.

2.1.7 Conversão para tons de cinza

- Converte uma imagem colorida (PNM P3) para escala de cinza (PNM P2). A conversão é realizada calculando a média ponderada dos componentes RGB para cada pixel.

2.2 Fluxo de execução

- O programa inicia solicitando ao usuário o nome do arquivo de imagem a ser processado.
- O menu de opções é exibido, oferecendo as operações listadas anteriormente.
- Após a escolha de uma operação, o usuário insere os parâmetros necessários, como fatores de brilho ou contraste, se aplicáveis.
- A operação é realizada e a imagem modificada é salva em um novo arquivo com o nome fornecido pelo usuário.
- O programa retorna ao menu principal, permitindo a escolha de uma nova operação ou a finalização do programa.

2.3 Exemplo de uso

Exemplo de fluxo de execução do programa:

1. O usuário insere o nome da imagem PNM original.
2. Escolhe a opção de **Clareamento**.
3. Insere o fator de clareamento.
4. O programa processa a imagem e salva o arquivo modificado.
5. O usuário pode então visualizar a nova imagem gerada ou continuar aplicando outras operações.

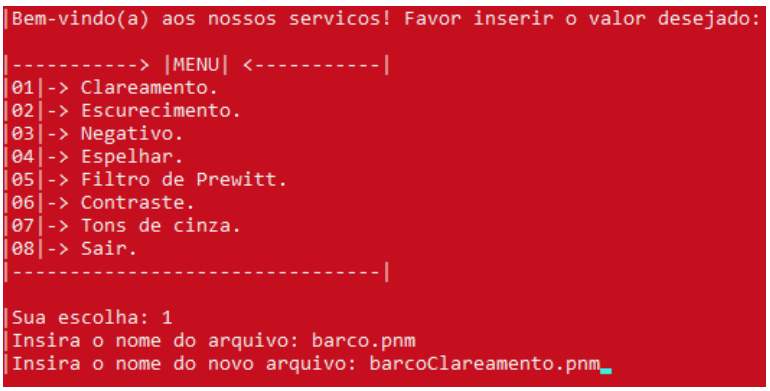

Figura 4: Exemplo funcionamento para filtro de clareamento

3. Estrutura de dados e funções auxiliares

3.1 Estrutura de dados *Cor*

Para representar os pixels de uma imagem PNM, foi definida a estrutura Cor, que armazena os valores RGB de cada pixel (para imagens coloridas) ou utiliza um único valor replicado para os três componentes (para imagens em tons de cinza). A estrutura foi projetada para permitir a manipulação eficiente das cores e intensidades das imagens processadas.

Cabe salientar que a utilização dessa estrutura unificada, tanto para imagens coloridas quanto para imagens em tons de cinza, foi uma escolha feita em prol de uma engenharia mais robusta e simplificada do programa. Isso evita a necessidade de criar estruturas diferentes para cada tipo de imagem, o que poderia aumentar a complexidade do código. Assim, por exemplo, na declaração Cor *img[MAXALTURA][MAXLARGURA]*, para imagens em tons de cinza, temos a seguinte atribuição:

img[i][j].r = img[i][j].g = img[i][j].b;

Dessa forma, mesmo em imagens em tons de cinza, os três componentes (r, g, b) são preenchidos com o mesmo valor, garantindo que a estrutura *Cor* seja utilizada de maneira consistente em todo o programa.

```
//struct com informacoes do rgb
struct Cor {
    unsigned char r;
    unsigned char g;
    unsigned char b;
};
```

Figura 5: Definição da estrutura struct RGB.

3.2 Funções auxiliares

- ***clearScreen()***: Limpa o terminal para melhorar a visualização do menu, garantindo uma interface mais limpa e organizada. Implementado tanto para sistemas windows como Linux, MacOS, Unix e Ubuntu.
- ***arqRead()***: Realiza a leitura dos arquivos PNM, armazenando os valores de pixel em uma matriz, além de capturar informações como tipo de imagem (P2 ou P3), dimensões e o valor máximo de cor.
- ***arqRecord()***: Salva a imagem processada em um novo arquivo PNM, mantendo o formato adequado (P2 ou P3) e preservando as propriedades do arquivo original.

3.3 Constantes definidas

- *MAX_PIXEL*: Define o valor máximo para os pixels da imagem (255).
- *MAXALTURA* e *MAXLARGURA*: Limites para as dimensões das imagens processadas (500x500 pixels).

- Matrizes convolucionais `Gx` e `Gy`: Usadas na operação de filtro de Prewitt para calcular os gradientes horizontais e verticais, respectivamente.

4. Implementação técnica

4.1 Bibliotecas utilizadas

O programa utiliza as seguintes bibliotecas da linguagem C++:

- `<iostream>`: Para entrada e saída de dados.
- `<fstream>`: Para leitura e gravação de arquivos.
- `<cstdlib>`: Para comandos do sistema operacional, como a limpeza da tela.
- `<string>`: Para manipulação de strings.
- `<limits>`: Para operações de controle de limites numéricos.
- `<cmath>`: Para operações matemáticas necessárias nos cálculos de processamento de imagem.

4.2 Algoritmos de processamento de imagem

Cada operação foi implementada seguindo algoritmos específicos de processamento de imagem, com destaque para:

- **Filtro de Prewitt**: Implementado por meio de convolução de matrizes, utilizando gradientes para realçar bordas na imagem.
- **Ajuste de contraste**: Implementado calculando o brilho médio da imagem e ajustando os valores de pixel em torno desse ponto médio, conforme o fator inserido pelo usuário.

5. Observações e restrições

5.1 Estrutura do programa

- O código precisa ser mais robusto ao lidar com erros de entrada do usuário, como valores fora do intervalo permitido e tipos de imagem não suportados (como P1, P4 e P5).
- O programa não pode manipular imagens maiores que 500x500 pixels devido ao uso de constantes como `MAXALTURA` e `MAXLARGURA`.

5.2 Portabilidade

- Dependendo da IDE ou ambiente de execução, funções como `system("pause")`, `system("color")` e `system("cls")/("clear")` podem não funcionar adequadamente. Essas funções dependem do sistema operacional e do terminal em uso, podendo apresentar comportamentos inesperados em IDEs como Visual Studio Code ou CLion. Para maior portabilidade, na elaboração de versões futuras o autor do código (e também deste relatório) irá considerar alternativas que não dependam dessas funções específicas.

6. Conclusão

O desenvolvimento deste programa de processamento de imagens em C++ proporcionou uma compreensão mais profunda das operações básicas e avançadas em manipulação de imagens, como clareamento, escurecimento, negativo, espelhamento, filtro de Prewitt, ajuste de contraste e conversão para tons de cinza. Através da implementação dessas funcionalidades, foi possível explorar e aplicar conceitos de programação, processamento digital de imagens e manipulação de arquivos.

A estrutura do programa foi projetada para ser flexível e eficiente, utilizando a estrutura *Cor* para representar pixels tanto em imagens coloridas quanto em tons de cinza. As funções auxiliares implementadas permitem a leitura e gravação de imagens no formato PNM, enquanto os filtros e ajustes aplicados demonstraram a eficácia dos algoritmos escolhidos.

7. Anexos

Figura 6: Imagem original, pasto cinza.

Figura 7: Imagem original, pasto colorido.

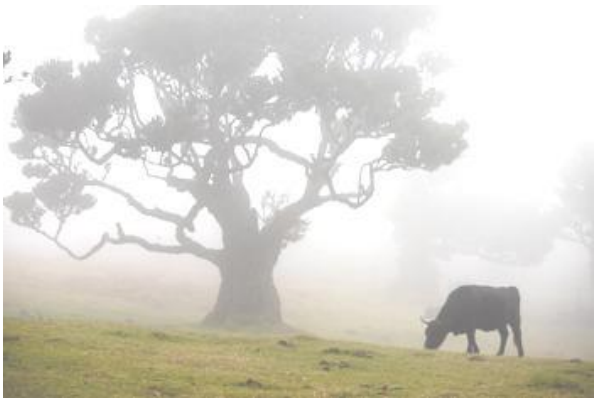

Figura 8: Imagem original, pasto cinza, filtro de clareamento.

Figura 9: Imagem original, pasto colorido, filtro de clareamento.

Figura 10: Imagem original, pasto cinza, filtro de escurecimento.

Figura 11: Imagem original, pasto colorido, filtro de escurecimento.

Figura 12: Imagem original, pasto cinza, filtro de espelhamento.

Figura 13: Imagem original, pasto colorido, filtro de espelhamento.

Figura 14: Imagem original, pasto cinza, filtro de imagem negativa.

Figura 15: Imagem original, pasto colorido, filtro de imagem negativa.

Figura 16: Imagem original, pasto cinza, filtro de Prewitt.

Figura 17: Imagem original, pasto colorido, efeito de Prewitt.