

Documentação para o Trabalho Prático de AEDS 2

Autor: Vítor Gabriel Reis Caitité

Prof. Erickson R. Nascimento

1. Introdução:

O objetivo deste trabalho é implementar e aplicar a operação de convolução em imagens digitais para encontrar suas derivadas no formato PGM. Esse formato armazena uma imagem em tons de cinza.

Primeiramente é necessário entender o conceito de convolução. Essa operação é o tratamento de uma matriz por outra chamada “núcleo” (kernel).

O filtro de Matriz de Convolução usa uma matriz primária, que é a imagem a ser tratada. Essa figura é tratada como uma matriz de pixels, que no caso desse trabalho, todos esses são números que estão dentro do intervalo [0, 255]. O núcleo a ser usado vai depender do efeito desejado. Nesse processo especificamente é utilizado um kernel de tamanho 3x3. Vale citar que é possível a aplicação de diversos efeitos em imagens apenas com núcleos desse tamanho.

O filtro analisa cada pixel da imagem sucessivamente. Para cada um deles, que pode chamá-los de “pixel inicial”, o valor desse pixel será o somatório da multiplicação dele e dos 8 pixels ao redor do mesmo pelo valor correspondente na matriz núcleo. Esses nove valores são adicionados e se tornam o valor final daquele pixel. Esse processo foi demonstrado abaixo:

35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

 \times

	0	1	0	
	0	0	0	
	0	0	0	

 $=$

		42		

A única diferença desse processo descrito acima e o aplicado durante o processo convolução para se obter a derivada de uma imagem é que a matriz original não tem seus termos multiplicados exatamente pela posição correspondente no kernel, pois ao se observar a fórmula da convolução percebe-se que o processo realizado por ela é semelhante a multiplicar os termos da matriz original pelos termos correspondentes da matriz kernel rotacionada 180°, como ilustrado:

5	5	4	7	0	1
3	2	4	9	1	2
4	6	5	8	5	2
4	5	5	8	1	3
5	5	4	7	0	1

Imagem

a	b	c
d	e	f
g	h	i

kernel

i=5	h=5	g=5						
f=5	e=5	d=5	4	7	0	1		
c=3	b=3	a=2	4	9	1	2		
			4	6	5	8	5	2
			4	5	5	8	1	3
			5	5	4	7	0	1

Convolução

2. Implementação:

Estrutura de Dados

Para a implementação do trabalho foi criado um Tipo Abstrato de Dados PGM com os seguintes campos:

- int c - Número de colunas da imagem
- int l - Número de linhas da imagem
- unsigned char máximo - Valor máximo para cada pixel
- unsigned char **imagem - ponteiro de ponteiro para os pixels da imagem (matriz)

Funções e Procedimentos

O TAD desenvolvido possui as seguintes funções:

PGM*LerPGM(char*entrada): Essa função primeiramente aloca memória para o variável do tipo da estrutura de dados desenvolvida a qual foi dada o nome de img. Então ela lê de um arquivo (.pgm) alguns dados importantes como o números de linhas e de colunas da matriz de pixels e também o mais valor que cada pixel pode possuir. A partir desses valores ela chama uma função que aloca memória para a matriz de pixels à qual existe na estrutura um ponteiro de ponteiro associado a ela.

O próximo passo realizado é chamar a função fscanf repetidamente para assim ler e armazenar em img->imagem os valores da matriz de pixels lida.

Por fim, essa função fecha o arquivo de onde se leu todos os dados e retorna um endereço onde foi alocado memória e armazenado os elementos.

void Convolucao(PGM* img, char kernel, PGM* saida):** A função recebe como parâmetros o endereço do local onde estão todos os dados da imagem original, o endereço do kernel e o endereço onde ele deve colocar a imagem convoluída. Primeiramente foi necessário declarar e alocar memória dinamicamente para PGM* img_aumentada (que irá receber as informações da imagem original, porém com os pixels das extremidades replicados). Além disso também foi alocado memória para a matriz dos pixels dentro dessa estrutura.

Feito isso, também são realizados os mesmos passos descritos acima para PGM* saída, que após ter sido realizada a convolução possuirá os dados da nova imagem.

Por fim realiza-se a operação propriamente dita que consiste em analisar cada pixel da imagem sucessivamente. Para cada um deles, o valor desse pixel será o somatório a multiplicação dele e dos 8 pixels ao redor do mesmo pelo valor correspondente, de acordo com a fórmula de convolução, na matriz núcleo. Esses nove valores são adicionados e se tornam o valor final daquele pixel. No

programa essa sequência de operações é realizada de acordo com a fórmula descrita abaixo:

$$h[x, y] = f[x, y] * g[x, y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$

OBS.: Foi criada uma nova matriz de pixels com as extremidades replicadas a partir da matriz original, antes da convolução, para evitar que a nova imagem (convoluída) seja menor do que a imagem de entrada. Ao final dessa função a memória que armazenava todos os dados da matriz aumentada é desalocada.

unsigned char alocacao(int lines, int columns):** Essa função é responsável por alocar memória para as matrizes de pixels (por isso a função é do tipo unsigned char**) utilizadas durante o funcionamento do programa. Ela recebe como parâmetros as linhas e colunas da matriz a ser alocada.

O processo realizado foi alocar um vetor de ponteiros, então percorrer esse vetor e alocar um vetor de unsigned char para cada posição dele. E no fim da função retorna-se o endereço para a matriz alocada.

void liberarPGM(PGM* img): Procedimento que libera a memória alocada para a variável do tipo da estrutura implementada e para todos os campos que ela possui (inclusive para a matriz de pixels).

Primeiramente libera-se todas as linhas dessa matriz, para assim desalocar também o vetor de ponteiro restante. Após isso bastou liberar a memória alocada para a variável do tipo da estrutura implementada.

char GeraKernel():** Essa função aloca a memória necessária e ainda gera o kernel utilizado para realizar a derivada de imagens. Como já se sabe o tamanho do kernel e seus dados não é necessário receber nada como parâmetro. Primeiramente a função aloca memória pra a matriz núcleo como já descrito acima e depois apenas o inicializa atribuindo seus respectivos valores.

void liberarKernel(char kernel):** Recebe como parâmetros o endereço onde está armazenada a matriz kernel e com isso tem o objetivo de desalocar a memória alocada para ela. Primeiramente libera-se todas as linhas desse núcleo (ou kernel), para assim desalocar também o vetor de ponteiro restante.

Programa principal: `int main(int argc, char *argv[])`: Primeiramente o programa principal, cria um ponteiro do tipo PGM (que inicialmente aponta para NULL) que irá apontar para os dados que serão lidos da imagem.pgm original, e cria também um ponteiro desse mesmo tipo (que inicialmente também aponta para null) mas que irá apontar para a imagem derivada.

Ainda nessa fase de declarações também foi declarado um ponteiro de ponteiro (**kernel). A matriz kernel foi iniciado por uma função GeraKernel chamada no main, e que retorna o endereço do local onde está a matriz kernel já iniciada.

Também foi chamada no main a função LerPGM que lê os dados do arquivo.pgm e também aloca a memória onde esses dados ficarão armazenada. Feito isso se fez necessário alocar dinamicamente a variável do tipo da estrutura criada que recebe a matriz convoluída.

Tendo assim, portanto, todas as memórias necessárias alocadas pôde-se chamar a função que realiza a convolução, passando 3 dados como parâmetros: o endereço de onde estão armazenados os dados da imagem original, o endereço da matriz kernel, e o endereço para a memória alocada para receber a matriz convoluída.

Uma vez feita a convolução os dados da imagem derivada já foram salvos, e falta apenas criar um arquivo para ser a nova imagem (convoluída). Para tal ato bastou-se chamar no main a função específica para realização dessa atividade, sendo essa a função SalvarPGM.

E por fim, restou-se desalocar as memórias alocadas dinamicamente. Para isso foi invocado o procedimento liberaPGM duas vezes para liberar todos os dados da imagem original e da imagem derivada. E também foi chamada a função liberaKernel que libera matrizes de ordem 3x3.

Organização do Código, Decisões de Implementação e Detalhes Técnicos

O código está dividido em três arquivos principais: Funcoes.c e Funcoes.h implementam o Tipo Abstrato de Dados enquanto o arquivo main.c implementa o programa principal.

Além das funções exigidas resolvi implementar outras funções para facilitar a execução do projeto, essas funções foram:

- `unsigned char** alocao(int lines, int columns)`
- `void liberarPGM(PGM* img)`
- `char** GeraKernel()`
- `void liberarKernel(char** kernel)`

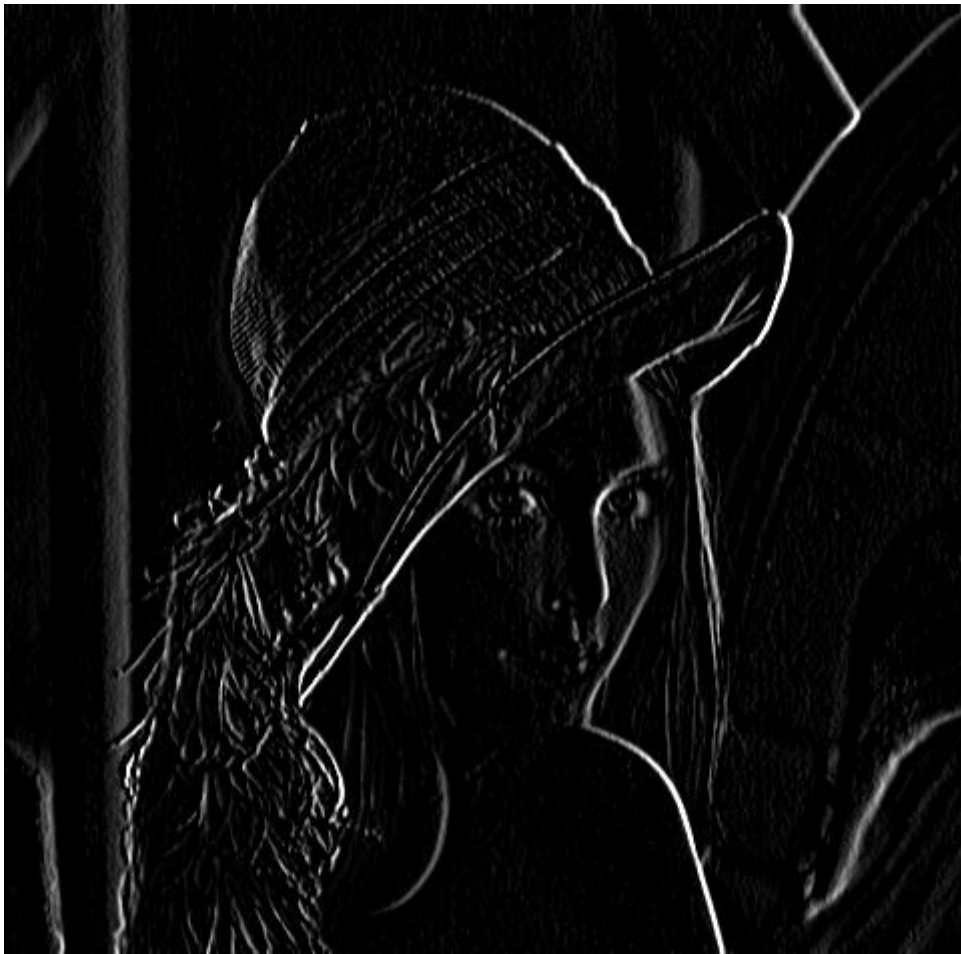
O compilador utilizado foi o gcc no sistema operacional Linux Ubuntu. Para executar o programa, basta compilar e então a partir da linha de comando abrir o programa e passar como argumentos o nome do arquivo com a imagem

original (imagem essa que deve estar no mesmo diretório do programa) e o nome do arquivo a ser criado contendo a imagem convoluída.

Exemplo: ./nome_do_programa lena.pgm ImagemConvoluida.pgm

3. Testes:

Vários testes foram realizados com o programa de forma a verificar o seu funcionamento. Utilizando-se a imagem lena.pgm encontrou-se exatamente o resultado esperado mostrado abaixo:



À título de curiosidade e também para aprofundar um pouco mais no tema foram aplicadas convoluções também para outros kernels e se obteve diversos resultados, os quais também estão anexados na página seguinte.



(1 1 1,0 0 0,1 1 1)



(0 1 0,1 1 1,0 1 0)



(-1 -2 -1,0 0 0,1 2 1)



(-1 -1 -1, -1 9 -1, -1 -1 -1)

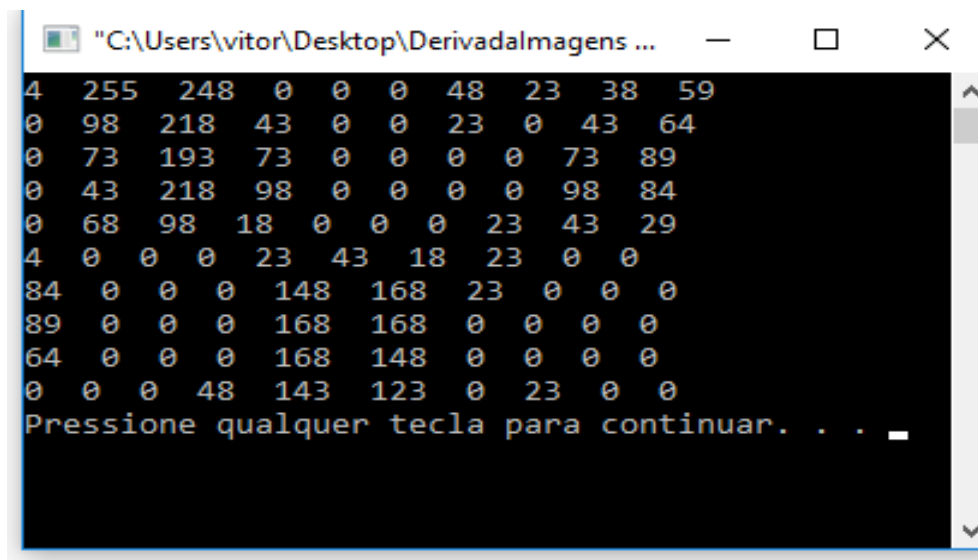


(0 -1 0,-1 5 -1,0 -1 0)



(1 1 1,-1 -1 -1,1 1 1)

Para testar ainda mais o código o utilizei para fazer a convolução da imagem teste e mostrar na tela a matriz convoluída para assim poder compara-la com o resultado passado na descrição do TP. Como pode-se ver abaixo o resultado foi exatamente o esperado:



```
"C:\Users\vitor\Desktop\DerivadaImagens ..."
4 255 248 0 0 0 48 23 38 59
0 98 218 43 0 0 23 0 43 64
0 73 193 73 0 0 0 0 73 89
0 43 218 98 0 0 0 0 98 84
0 68 98 18 0 0 0 23 43 29
4 0 0 0 23 43 18 23 0 0
84 0 0 0 148 168 23 0 0 0
89 0 0 0 168 168 0 0 0 0
64 0 0 0 168 148 0 0 0 0
0 0 0 48 143 123 0 23 0 0
Pressione qualquer tecla para continuar. . .
```

4. Conclusão:

A partir de tudo mostrado acima, pode-se concluir que a implementação do trabalho transcorreu sem maiores problemas e os resultados ficaram dentro do esperado. A realização dessa atividade além de gerar maior conhecimento no tema, ainda foi importante para afixar ainda mais a parte da matéria relacionada a alocação dinâmica e manipulação de imagens.

A principal dificuldade se deu pelo fato de inicialmente eu estar implementando o código no sistema operacional Windows e durante o período de testes ter mudado para Linux, com isso foi necessário a realização de algumas adequações no código que causaram uma certa dificuldade, porém essa pôde ser superada com sucesso.

Referências

[1] Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C, 2ª Edição, Editora Thomson, 2004.

[2] <https://docs.gimp.org/2.8/pt_BR/plugin-convmatrix.html> Acesso em: 16/04/17

Anexos:

Listagem dos programas:

- Funcoes.h
- Funcoes.c
- main.c