

MARIANA MEIRELES GONTIJO

DOCUMENTAÇÃO TP0

April 21, 2017

ID: 2015079208

Universidade Federal de Minas Gerais
Departamento de Engenharia Elétrica

1. INTRODUÇÃO

O trabalho consiste em utilizar o método de convolução de matrizes para encontrar mudanças abruptas de cores, que geralmente são as fronteiras de objetos. Para isso precisa-se utilizar uma matriz denominada kernell, que deriva cada pixel da matriz de imagem original, deixando em evidência os contornos dos objetos.

2. IMPLEMENTAÇÃO

0.1 ESTRUTURA DE DADOS

O programa consiste ao todo em três arquivos, o matrix.h, que é a biblioteca onde armazenamos estruturas de dados e cabeçalhos de funções, o matrix.c que é onde os dados das estruturas do matrix.h são manipulados e as funções desenvolvidas e o main.c que contém as instruções de como o programa deve usar as informações contidas no matrix.c.

```
typedef struct {  
    int c;                /*numero de colunas*/  
    int l;                /*numero de linhas*/  
    unsigned char maximo; /*valor max de um numero*/  
    unsigned char **imagem; /*numeros que compoe a imagem*/  
} PGM;
```

```
PGM *LerPGM (char* entrada);  
void Convolucao (PGM *img, char **kernell, PGM *saida);  
void SalvarPGM (PGM *img, char *saida);
```

O struct PGM armazena os dados das imagens a serem convolucionadas pelo programa. Já as funções tem suas especificidades, e serão apresentadas e em seguida explicadas com mais detalhes na próxima sessão.

- A função PGM *LerPGM (char* entrada) recebe o nome do arquivo de entrada e retorna um PGM idêntico ao de arquivo de entrada que será manipulado pelo programa.
- A função void Convolucao (PGM *img, char **kernell, PGM *saida) recebe dois arquivos do tipo PGM e uma matriz do tipo char que será a responsável por realizar as derivadas necessárias para a convolução do arquivo.
- A função void SalvarPGM (PGM *img, char *saida) recebe o arquivo já convolucionado e o grava em um arquivo de saída.

0.2 FUNÇÕES E PROCEDIMENTOS

PGM *LerPGM (char* entrada)

A função recebe uma string de caracteres que indica qual o nome do arquivo a ser aberto. Depois abre o arquivo e cria uma variável x do tipo PGM para armazenar os dados do arquivo. A medida que o programa lê as variáveis do arquivo ele alocar dinamicamente a variável x para que ocupe só o espaço necessário na memória.

```
/*aloca dinamicamente dados simples da PGM*/
x.c = malloc(sizeof(int));
x.l = malloc(sizeof(int));
x.maximo = malloc(sizeof(unsigned char));

/*atribui valores para os dados simples da PGM*/
fscanf(fin , "%d" , &x.c);
fscanf(fin , "%d" , &x.l);
fscanf(fin , "%d" , &x.maximo);

/*aloca dinamicamente a matriz da PGM*/
x.imagem = malloc(x.l * sizeof(unsigned char *));

for(i = 0; i < x.l; i++)
    x.imagem[i] = malloc(x.c * sizeof(unsigned char));

/*atribui os valores do arquivo de entrada para a matriz
x.imagem*/
for (i = 0; i < x.l; i++) {
    for (j = 0; j < x.c; j++) {
        fscanf(fin , "%d" , &x.imagem[i][j]);
        printf("matrix.c: na posicao [%d][%d]"
            " encontra-se o valor %d\n" , i , j , x.imagem[i][j]);
    }
}
```

Como pode ser visto no código acima essa parte foi feita passo a passo, decidi por essa implementação para que eu pudesse entender melhor o que estava acontecendo, fazer desse modo deixou o processo mais claro pra mim e facilitou na hora de procurar e entender bugs.

```
PGM *y;  
y = malloc(sizeof(PGM));  
y = &x;
```

Depois de todos os dados alocados na variável x, o programa cria uma variável *y to tipo ponteiro para PGM a aloca dinamicamente, dessa vez a alocação foi concisa, já que já havia entendido o processo.

void Convolucao (PGM *img, char **kernell, PGM *saida)

A função recebe como argumentos um ponteiro para PGM de entrada chamado *img, um ponteiro para PGM de saída chamado *saida e uma matriz tipo char chamada kernell.

Essa é a função que realiza o processo de convolução. Para realizar esse processo é necessário fazer uma multiplicação de matrizes entre a matriz da imagem e a matriz do kernell. O kernell utilizado tem tamanho 3x3 e para multiplicar essas matrizes sem que a imagem original sofresse perda de tamanho foi necessário tomar um cuidado especial com suas bordas. Para isso foi criada uma variável **matrix que recebeu como tamanho o tamanho original da matrix presente em img somado de dois.

	a_{00}	a_{01}	
	a_{10}	a_{11}	

Na imagem acima uma matriz exemplo de 2x2 representa a matriz original, `img->imagem`. As bordas adicionais foram criadas na `**matrix`, matriz presente apenas nesta função. Após a manipulação da matriz por meio do código, vê-se o resultado na imagem abaixo.

a_{11}	a_{11}	a_{12}	a_{12}
a_{11}	a_{11}	a_{12}	a_{12}
a_{21}	a_{21}	a_{22}	a_{22}
a_{21}	a_{21}	a_{22}	a_{22}

Os valores da matriz original foram replicados em todas as bordas da tabela, a fim de que a imagem continuasse do mesmo tamanho da original. Preferi implementar do meio para as extremidades da matriz, atribuindo primeiro os valores do meio, depois os valores das extremidades e depois os valores das quinas da matriz. Preferi essa implementação para não ter que me preocupar com algo sendo sobreescrito, já que a implementação trata do mais "grosso" para o mais "fino" e sobreescreve apenas os dados que não deveriam estar lá. Usei `fors` e atribuição direta para tal.

Depois a função aloca a PGM de saída, da mesma forma como aloquei a de entrada no começo, novamente, acredito que essa implementação mais detalhada me deu mais poder sobre o código, permitindo que eu visse exatamente o que estava acontecendo e possibilitando testar coisas diferentes. A matriz do kernell recebe seus respectivos valores para realizar a convolução.

A convolução em si é feita pelo seguinte código:

```
for (i = 1; i < img->l; i++) {
    for (j = 1; j < img->c; j++) {
        for (a = i - 1; a < (i) + 2; a++){
            if (k1 > 2) k1 = 0;
            k1++;

            for (b = j - 1; b < (j) + 2; b++) {
                if (k2 > 2) k2 = 0;

                soma = soma + matrix[a][b] * kernell[k1][k2];
                printf("(%d) matrix[%d][%d] * "
                    "(%d) kernell[%d][%d] = %d\n",
                    matrix[a][b], a, b, kernell[k1][k2], k1,
                    k2, soma);

                k2++;
            }

            soma = 0;
            if (soma < 0) soma = 0;
            if (soma > 255) soma = 255;
            saida->imagem[i-1][j-1] = soma;
            printf("SAIDA: %d\n", soma);
            soma = 0;
            k1 = -1;
            k2 = 0;
        }
    }
}
```

Os dois primeiros for percorrem a matriz `**matrix`, enquanto os dois últimos são responsáveis pela multiplicação da `**matrix` pelo `**kernell`. Os dois últimos for possuem um ponto de início e parada iterativos porque é importante relacionar os índices da `**matrix` enquanto a multiplicação pelo `kernell` é feita, esses índices iterativos garantem que a matriz continue se movendo e que o `kernell` multiplique partes diferentes dela toda vez. Os if ao final do código fazem os números após a convolução respeitarem o limite $0 < x < 255$. A variável `soma` contém o valor das somas das multiplicações dos valores ela atribui esses valores à PGM de saída.

void SalvarPGM (PGM *img, char *saida)

Essa função é responsável simplesmente por receber a PGM da matriz que foi derivada e atribuir a um arquivo de saída nomeado pelo usuário. A função abre o arquivo no modo de escrita e copia as informações presentes na informação para o arquivo.

0.3 PROGRAMA PRINCIPAL

A função main recebe do usuário os nomes de arquivo de entrada e saída. Declara as variáveis que serão usada no código e usa as funções explicadas na sessão anterior, abre o arquivo, faz uma cópia dele, utiliza o método de convolução e depois salva em outro arquivo.

3. RESULTADOS

Foi utilizado um exemplo menor nos testes enquanto o programa era construído. Contudo, os resultados que eles apresentam são irrelevantes comparados à convolução da imagem lena.pgm.



Acredito que resultados satisfatórios foram obtidos nessa convolução, visto que as bordas da imagem estão em evidência, em relação ao restante da imagem.

Contudo, com a imagem do carro os testes não foram tão satisfatórios. Após mudar o seguinte trecho de código, de receber int para receber char, algo não funciona bem. O trecho a seguir se encontra na função PGM *LerPGM (char* entrada) em matrix.c.

```
/*atribui os valores do arquivo de entrada para a matriz
x.imagem*/
for (i = 0; i < x.l; i++) {
    for (j = 0; j < x.c; j++) {
        fscanf(fin, "%c", &x.imagem[i][j]);
        printf("matrix.c: na posicao [%d][%d]"
            " encontra-se o valor %d\n", i, j, x.imagem[i][j]);
    }
}
```

O que obtenho como resposta é a imagem que segue, uma convolução sombreada da superposição de outra convolução.

Imaginei algumas teorias para explicar o porquê disso acontecer, mas a mais plausível é que há algo de errado com o modo como o programa lida com caracteres. Infelizmente,



isso só ficou óbvio pra mim no final do código, imaginei que a imagem carro.pgm teria o mesmo formato da imagem lena.pgm, que como demonstrado acima não resultou em nenhum problema. Como já havia desenvolvido todo o código não consegui chegar à raiz do problema, editei algumas partes do código mas, não consegui melhorar essa convolução, apenas piorar.

3. CONCLUSÕES

Minha maior dificuldade foi lidar com a alocação dinâmica dos dados, já que nunca havia feito isso antes com uma estrutura de dados à parte, nesse caso a biblioteca matrix.h. Outra parte que causou dificuldade no trabalho foi criar o código e a matriz para a convolução, criar a matriz foi trabalhoso e me pergunto se esse foi o jeito mais eficiente de implementá-la, acredito que com as ferramentas das quais temos conhecimento até o momento esse foi um dos melhores jeitos, contudo. Como conclusão, acredito que mesmo que não tenha sido de forma ótima, o Trabalho Prático que produzi cumpriu com parte considerável das especificações.

Aprendi a lidar com alocação dinâmica e estou mais segura do uso de ponteiros. De forma geral, foi um exercício bastante útil.

Compilando o programa:

Para rodar o programa, basta digitar make no terminal e depois ./edit . É necessário que o usuário forneça o nome da imagem de entrada com a extensão do arquivo e o nome da imagem de saída.

REFERÊNCIAS

cpreference.com

wikipedia.com

stackoverflow.com

gnu.org/software/make/manual/make.html#Reading

overleaf.com

cplusplus.com/

Slides da aula do Prof. Erickson