Documentação Trabalho Prático 0

20/04/2017

João Moreno
2015058510

Introdução:

O problema dado tem como objetivo aplicar a operação de convolução em uma imagem digital. Mais especificamente, utilizaremos a convolução para calcular a derivada de uma imagem, que nada mais é que um tratamento de imagem que busca encontrar mudanças bruscas de cor. Essas mudanças geralmente representam fronteiras entre objetos e/ou o fundo da imagem.

Implementação:

Para implementação desse trabalho foi criado um tipo abstrato de dados para tratar a imagem. A imagem é do tipo PGM de no máximo 256 cores. E as funções da TAD são voltadas à convolução, abertura e salvamento dos arquivos.

O tipo abstrato de dados possuí 4 funções principais, LerPGM, Convolucao, Inicia_PGM_Saida e SalvarPGM; elas serão discutidas a seguir.

- A implementação da função LerPGM pode ser dividida em 4 partes. A primeira é responsável por abrir o arquivo e checar se ele possui no cabeçalho o escrito P2, o que determina se é uma imagem PGM.

  A segunda lê as informações do cabeçalho sobre a imagem e salva na estrutura de dados.

  A terceira copia a imagem do arquivo para a estrutura de dados.

  E a quarta prepara a imagem que está na estrutura de dados para ser processada, na convolução.

```
PGM *LerPGM (char* entrada) {
    // Abrir o arquivo
    FILE *imagem;
    imagem=fopen(entrada, "r");
    if (!imagem) {
        printf("Erro na abertura do arquivo.\n");
        exit (1);
    }
    char *c;
    c=(char *)malloc(3*sizeof(char));
    fscanf(imagem, "%s", c);
    c[2] = '\0';
    if (strcmp(c, "P2") != 0){
        printf("\nO Arquivo nao e PGM.\n");
    }
    free(c);

    // Iniciar PGM_Entrada
    PGM *PGM_Entrada=malloc(sizeof(PGM));
    fscanf (imagem, "%d", &PGM_Entrada->c);
    fscanf (imagem, "%d", &PGM_Entrada->l);
    fscanf (imagem, "%hhu", &PGM_Entrada->maximo);
    unsigned char **MatrizEntrada=(unsigned char **)malloc((PGM_Entrada->l+2)*sizeof(char *));
    int ii;
    for (ii=0; ii < PGM_Entrada->l+2; ii++) {
        MatrizEntrada[ii]=(unsigned char *)malloc((PGM_Entrada->c+2)*sizeof(char));
    }
    PGM_Entrada->imagem=MatrizEntrada;

    // Copiar imagem em PGM_Entrada.imagem
    int jj;
    for(ii=1; ii < PGM_Entrada->l+1; ii++) {
        for (jj=1; jj < PGM_Entrada->c+1; jj++) {
            if (fscanf(imagem,"%hhu", &PGM_Entrada->imagem[ii][jj]) != 1) {
                printf("\nErro ao copiar o centro da imagem.\n");
                exit (2);
            }
        }
    }
    // Copiar bordas PGM_Entrada.imagem
    for(ii=1; ii < PGM_Entrada->l+1; ii++) {
        PGM_Entrada->imagem[ii][0]=PGM_Entrada->imagem[ii][1];                          // Copiar a borda esquerda
da imagem
    }
    for(ii=1; ii < PGM_Entrada->l+1; ii++) {
        PGM_Entrada->imagem[ii][PGM_Entrada->c+1]=PGM_Entrada->imagem[ii][PGM_Entrada->c]; // Copiar a borda direita
da imagem
    }
    for(ii=0; ii < PGM_Entrada->c+2; ii++) {
        PGM_Entrada->imagem[0][ii]=PGM_Entrada->imagem[1][ii];                          // Copiar a borda inferior
da imagem
    }
    for(ii=0; ii < PGM_Entrada->c+2; ii++) {
        PGM_Entrada->imagem[PGM_Entrada->l+1][ii]=PGM_Entrada->imagem[PGM_Entrada->l][ii]; // Copiar a borda superior
da imagem
    }
    fclose (imagem);
    return PGM_Entrada;
}
```

- A função Convolucao é o núcleo do programa. Ela é responsável por fazer todo o cálculo matemático da derivada parcial da imagem. Pode ser dividida em duas partes.

  A primeira parte é responsável por executar os cálculos e a segunda é responsável por desalocar as estruturas que não serão mais necessárias para o programa, tais como a matriz kernel e a imagem de entrada.

```
void Convolucao(PGM *PGM_Entrada, char **kernel, PGM *PGM_Saida) {
    // Fazer a convolucao
    int jj, ii;
    for (ii=1; ii < PGM_Entrada->l+1; ii++) {
        for (jj=1; jj < PGM_Entrada->c+1; jj++) {
            int val =
            (int) PGM_Entrada->imagem[ii-1][jj-1]*kernel[2][2] + PGM_Entrada->imagem[ii-1][jj]*kernel[2][1] +
PGM_Entrada->imagem[ii-1][jj+1]*kernel[2][0] +
                  PGM_Entrada->imagem[ii  ][jj-1]*kernel[1][2] + PGM_Entrada->imagem[ii  ][jj]*kernel[1][1] +
PGM_Entrada->imagem[ii  ][jj+1]*kernel[1][0] +
                  PGM_Entrada->imagem[ii+1][jj-1]*kernel[0][2] + PGM_Entrada->imagem[ii+1][jj]*kernel[0][1] +
PGM_Entrada->imagem[ii+1][jj+1]*kernel[0][0] ;
            if(val > 255) val = 255;
            else if(val < 0) val = 0;
            PGM_Saida->imagem[ii-1][jj-1] = val;
        }
    }
    // Desalocar PGM_Entrada e kernel
    for (ii=0; ii < PGM_Entrada->l+2; ii++) {
        free(PGM_Entrada->imagem[ii]);
    }
    free(PGM_Entrada);
    for (ii=0; ii < 3; ii++) {
        free(kernel[ii]);
    }
    free(kernel);
}
```

- A função IniciaPGM_Saida somente cria uma estrutura do tipo PGM para armazenar as informações que irão ser geradas pela saída da função Convolucao.

```
PGM *IniciaPGM_Saida(PGM *PGM_Entrada) {
    // Iniciar PGM_Saida
    PGM *PGM_Saida=malloc(sizeof(PGM));
    PGM_Saida->c=PGM_Entrada->c;
    PGM_Saida->l=PGM_Entrada->l;
    PGM_Saida->maximo=PGM_Entrada->maximo;
    unsigned char **MatrizSaida=(unsigned char **)malloc(PGM_Saida->l*sizeof(char *));
    int ii;
    for (ii=0; ii < PGM_Entrada->l; ii++) {
        MatrizSaida[ii]=(unsigned char *)malloc(PGM_Entrada->c*sizeof(char));
    }
    PGM_Saida->imagem=MatrizSaida;
    return (PGM_Saida);
}
```

- A função SalvarPGM salva no arquivo de caminho indicado pela entrada do programa os valores da convolução que já foi calculada.

```c
void SalvarPGM(PGM *PGM_Saida, char* saida) {
    // Criar arquivo de saída
    FILE *saidaf;
    saidaf=fopen(saida, "w");
    if (!saidaf) {
        printf("Erro na abertura do arquivo de saida.\n");
        exit (8);
    }
    fprintf(saidaf, "P2\n%d %d %d\n", PGM_Saida->c, PGM_Saida->l, PGM_Saida->maximo);

    // Copiar PGM_Saida->imagem no arquivo
    int ii, jj;
    for(ii=0; ii < PGM_Saida->l; ii++) {
        for (jj=0; jj < PGM_Saida->c; jj++) {
            fprintf(saidaf,"%d ", PGM_Saida->imagem[ii][jj]);
        }
        fprintf(saidaf,"\n");
    }
    // Desalocando saidaf e PGM_Saida
    fclose(saidaf);
    for (ii=0; ii < PGM_Saida->l; ii++) {
        free(PGM_Saida->imagem[ii]);
    }
    free(PGM_Saida);
}
```

Resultados:

O programa está se comportando como o esperado. Não foi encontrado nenhum leak de memória pelo programa valgrind, usando o comando:

```
valgrind --leak-check=full -v ./a.out
```

Também não foi relatado nenhum erro nem warning durante a compilação pelo gcc, usando o comando:

```
gcc -Wall -Werror -Wextra main.c PGM.c
```

As saídas estão idênticas ao exemplificado.