INF01046 - Fundamentos de Processamento de Imagens

Projeto B: Realce de imagens - Filtros no domínio espacial

Cristiano Medeiros Dalbem - 173362

Instituto de Informática
Universidade Federal do Rio Grande do Sul

cristiano.dalbem@inf.ufrgs.br Data de entrega: 28 de outubro de 2009

28 de outubro de 2009

Resumo

Este projeto consiste na implementação de um programa de computador que aplique filtros de realce estudados em aula para imagens. O programa deve ser capaz de ler e salvar em disco imagens no formatoBMP, sem cores, e aplicar operações com máscaras (kernels) relativasa filtros do domínio espacial, ou seja, que trabalham diretamentecom os valores dos pixels nas tabelas em memória. São nativas do programatransformações básicas: Gaussiana, Laplace, Derivada Primeirae Média, mas é possível entrar com uma matriz arbitrária de valores aserem operados nas imagens. O programa também permite a soma deimagens, para análise dos filtros de aguçamento de bordas.

1 Discussão Técnica

Para a manipulação de imagens digitais com a linguagem de programação C++, escolhi utilizar a EasyBMP.

Na realidade não foi uma escolha compulsória, já que meus primeiros planos eram de criar um programa um pouco mais amigável que pudesse, no mínimo, ter uma integração com OpenGL. Planejava a criação de uma interface baseada em Windows, e não em Console, apenas, além de poder desenhar na tela. Por exemplo, um dos meus planos era a implementação de "sliders"que modificassem rapidamente os parâmetros das transformações, podendo assim que seus efeitos pudessem ser apreciados em tempo real com previews.

Minhas primeiras tentativas foram com a biblioteca DevIL, uma biblioteca que seria [supostamente] uma fácil interface entre bibliotecas gráficas. Ocorre que penei bastante para tentar extrair algumas ferramentas básicas desta biblioteca, o que não deveria ser problema, posto que já havia trabalhado com ela anteriormente. Ocorre que esta se mostrou mais complexa do que parecia, principalmente por conta de sua paupérrima documentação.

Finalmente optei, então, por utilizar uma biblioteca bem mais simples, com um poder bem menor do que o que minhas pretensões necessitavam. Esta é a EasyBMP, uma biblioteca implementada em C++ e faz muito bem o pouco a que se dedica: operar em cima de arquivos BMP.

1.1 Estrutura básica do programa

Em main.cpp temos apenas funções relativas a interface, como controle geral de como as funcionalidades do programa serão acessadas pelo usuário.

Abaixo, uma versão resumida da classe principal do programa que se encontra em classImage.h. Ela se dedica à imagem que está atualmente aberta.

```
class Imagem
{
    private:

                BMP original;
                BMP buffer;

    public:
        char imagePath[MAX_URL];
        int h, w;

        //TRANSFORMAÇÕES
```

```
        void convolucao_nxn(float *mask, int n);

        void laplaciano();
        void media();
        void gaussiano();
        void derivadaPrimeira();
        void customMask(int n);
        void soma();
};
```

Nela temos dois tipos BMP: uma que armazena a imagem original, alterando-a apenas para uma versão em preto-e-branco; outra que será, como diz o nome, um buffer, que estará armazenando as versões modificadas da imagem - graças a esse buffer é que se torna possível a aplicação sequenciada de diferentes filtros.
Todas as transformações são implementadas como funções deste objeto, sendo que só estas têm acesso às imagens propriamente ditas.

Há uma função generalizada para aplicação da convolução. Na realidade ela é limitada, pois neste trabalho resolvi utilizar apenas kernels quadrados (n por n) - generalizar para retângulos seria uma inserir uma complexidade desnecessária para o âmbito do aprendizado aqui envolvido.

Em classImage.cpp temos a implementação de todas funções relativas à classe Imagem.

1.2 As funções de aplicação de filtros

Para os filtros de aguçamento exigidos pela definição do projeto, bastou ao aluno a inicialização das matrizes com os valores de matrizes vistos em aula, sendo que sua razão de ser é basicamente a da definição de cada filtro e de seus objetivos particulares.

```
void Imagem::laplaciano()
{
    float mask[3][3] = { -1, -1, -1,
                         -1,  8, -1,
                         -1, -1, -1 };

    printMask((float*)mask,3);
    convolucao_nxn((float*)mask,3);
}
void Imagem::media()
```

```
void Imagem::gaussiano()
{
    float mask[3][3] = { 0.0625, 0.125 ,0.0625,
                         0.125,  0.25,  0.125,
                         0.0625, 0.125 ,0.0625 };

    printMask((float*)mask,3);
    convolucao_nxn((float*)mask,3);
}
```

Estes são exemplos típicos de implementações de funções para aplicação de um kernel pré-programado. O layout do código é exatamente o mesmo para os outros, variando os números. Já na função de kernel customizável, isto é, aquele cujos valores serão dados pelo usuários, teremos uma pequena interface para facilitar o processo, mas as chamadas essencialmente seguem sendo as mesmas:

- Definição da matriz que será utilizada na convolução;
- Impressão para o usuário da matriz sendo utilizada (já que ele apenas escolheu o filtro por nome);
- Chamada de função de convolução - como essas funções são definidas para o objeto Imagem, aqui só passamos informações a respeito do que faremos com a imagem, e não dados sobre o que ela já era.

Como visto em aula, a aplicação de uma máscara em uma matriz representando a imagem é feita através da operação de convolução, que no domínio espacial, equivale à multiplicação vetorial de cada pixel pela matriz correspondente à máscara.

```
int delta = floor(n/2);

for(int x=delta; x < w-delta; x++){
   for(int y=delta; y < h-delta; y++)
      for( int i = 0-delta;  i  <  n-delta; i++)
         for( int k = 0-delta;  k  <  n-delta; k++)
            matriz[x][y] = matriz[x][y] +original(x+i,y+k) * mask[i+delta][k+delta];
```

Esta é uma versão simplificada de o que se encontra no código do programa - é uma versão adaptada, limpando dos artifícios que eventualmente são necessários de serem utilizados dentro das especificidades de cada linguagem. Este trecho guarda a essência do que é a operação de convolução, e em muito se assemelha à implementação da mesma na maior parte das linguagens de programação. Inclusive, comparando-se com o que foi estudado em aula, em MathLab, os códigos são basicamente os mesmos.

2 Resultados

Algo que aprendi aqui é que o trabalho de outras pessoas de desenvolverem bibliotecas que nos permitam abstrair camadas que não nos interessam das implementações não necessariamente é repassada a nós. Num trabalho como este é muito importante a comunicação com outras pessoas com experiência na área para que se conheça bem que tipo de ferramentas são as mais adequadas para o tipo de tarefa que nos é designada, e que nos interessa

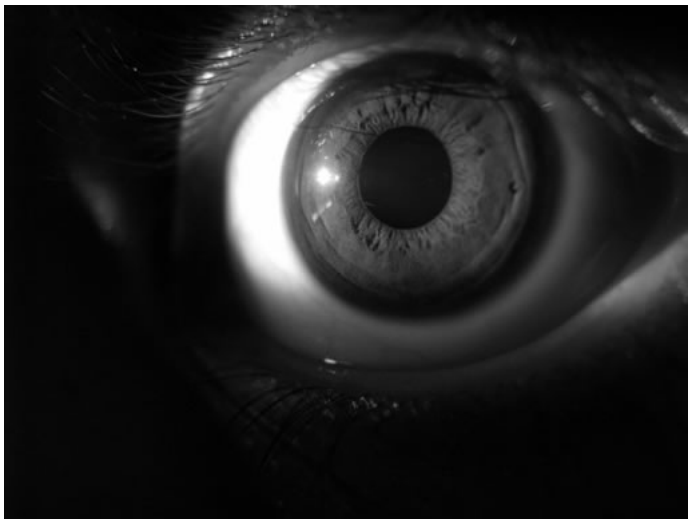
fazer. Por exemplo, o aluno poderia ter implementado completamente funções que permitam abrir arquivos BMP, já que é o formato de armazenagem de imagem mais simples que há. Porém, este não era o foco do trabalho, e nem mesmo o é da disciplina em si, no que concluímos ser esforço talvez não disperdiçado, mas mal condicionado.

Ainda que tenha havido certa complicação de minha parte com o assunto da biblioteca da manipulação, como já explicitado anteriormente, posso dizer que houve possibilidade de desenvolvimento no estudo da aplicação. Ferramentas como o MathLab são feitas para serem rápidas e poderosas, e isso só é possível porque o campo de atuação dela é bastante limitado - como já é bem explicitado no próprio nome. Assim, um mesmo programa para tratamento de imagens que consumiria algumas poucas linhas de código em MathLab pode ser tornar bastante complexa quando nos utilizamos de uma linguagem de programação tão genérica - e até de baixo nível, por que não dizer - como o C, ou mesmo o C++. Posso dizer que foi uma experiência relativamente desafiante, mas mais ainda gratificante, por poder implementar em uma linguagem com a qual me sinto mais familiarizado conceitos que pareciam genéricos demais para realmente funcionar numa aplicação própria que seja simples.

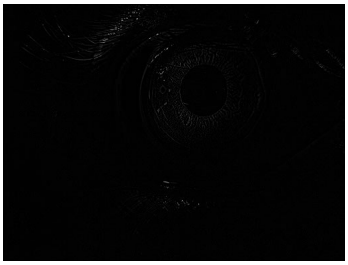
Já no que diz respeito aos resultados sobre a temática de manipulação de imagens, devo dizer que não houve tanto crescimento quanto no caso anterior. E isso se deve ao fato de o livro texto usar, para exemplificar a ação de cada filtro, as melhores imagens possíveis, isto é, em casos que os usos são mais óbvios e os resultados mais imediatos.

Apesar disso, eu como admirador da fotografia já pude me utilizar desse programa para pequenas utilizações do dia-a-dia, e neste escopo não podemos ignorar a utilidade dessas ferramentas, ainda que não tão poderosas. Um bom exemplo que me é recorrente é a necessidade, no redimensionamento de fotos para utilização na web; neste momento há uma grande perda de importância para detalhes que muitas vezes definem o interesse em uma imagem; neste momento se mostra imprescindível a utilização de filtros de aguçamento de bordas - chamado de "Sharpen"no famoso Photoshop, é um dos mais úteis em toda suíte.(ver Figura 1)

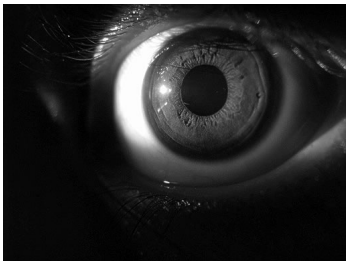
A figura mais ao topo é a original, a qual ficou extremamente sem graça ao ser reduzida a uma baixa resolução. O interesse aqui seria de alguma maneira fazer chamar mais a atenção nos detalhes da iris, que é o ponto de interesse da imagem. Por esta razão, essa é a imagem que escolhi para



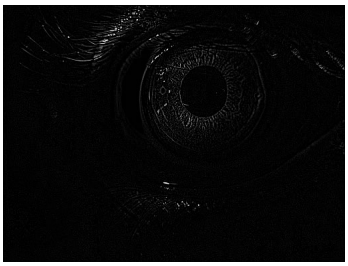
(a) Original



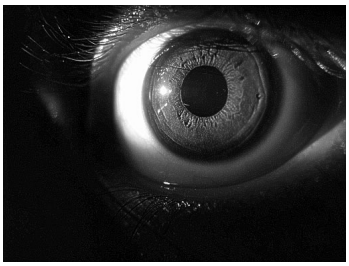
(b) Derivada primeira



(c) Derivada primeira + Original



(d) Laplace



(e) Laplace + Original

Figura 1: Demonstração de aguçamento de bordas

(a) Original

(b) Gaussiano

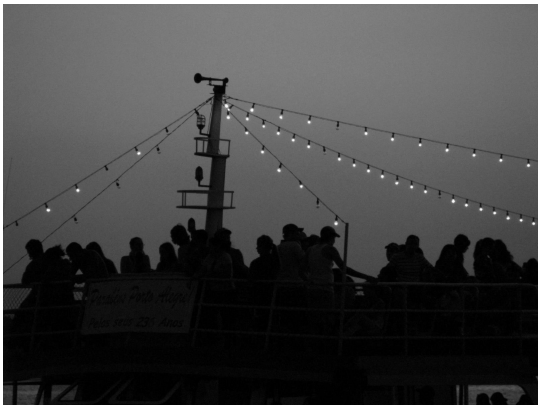
(c) Média

Figura 2: Demonstração de filtros passa-baixas

desmontrar melhor os efeitos dos filtros de aguçamento de borda. Na coluna da esquerda está a imagem diretamente processada pelos filtros Laplace e Derivada Primeira, respectivamente. Na coluna da direita, uma soma do resultado do filtro com a imagem original para termos de demonstração.

Para o teste dos filtros passa-baixas exigidos, escolhi uma fotografia com bastante ruído, adquirido naturalmente da alta sensibilidade ISO utilizada numa câmera digital de baixa qualidade. Vejamos se é um método interessante para aliviar o defeito (ver Figura 2).

A diferença entre os dois filtros para esta intensidade aplicada é quase imperceptível, ainda que o gaussiano pareça manter um pouco mais os detalhes sem perder na suavização de ruído. Porém, sabemos que estes não são



(a) Original

(b) Processado

Figura 3: Demonstração de High-Boost

métodos comumente aplicados a este problem, sendo úteis só e basicamente para se obter um borramento da imagem.

Um tipo de máscara que quis experimentar utilizando o recurso de Custom Mask foi o de High Boost. Para esta foto (ver Figura 3), foi utilizado um High Boost com os seguintes valores:

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 10 & -1 \\ -1 & -1 & -1 \end{matrix} \tag{1}$$

Para finalizar, uma brincadeira possível com o que foi desenvolvido é a combinação da soma com o borramento. Após algumas iterações no filtro

(a) Original (b) Processado

Figura 4: Demonstração de combinação de filtros

gaussiano, somamos o resultado com a imagem original, conseguindo um efeito "dreamy". (ver Figura 4)

3 Listagem

3.1 main.cpp

```
#include <iostream>
#include <string>
#include "conio2.h"

#include "classImage.h"

#define DEBUG cout<<__LINE__<<endl
#define WAIT cout<<"done! press any key to continue..."; getch()


using namespace std;



void menuLoad(Imagem *image)
{
    cout<<"enter path: ";
    char path[MAX_URL];
    cin >> path;
                while( !image->load(path) ) {
        cout << "Nao consegui abrir teu arquivo! Tenta de novo\a" << endl;
```

```
        cin >> path;
    }
    image->convertToGrayScale();
}

void menuSave(Imagem *image)
{
    char savename[MAX_URL];
    strncpy(savename,image->imagePath,strlen(image->imagePath)-3);
    savename[strlen(image->imagePath)-4]='\0';
    sprintf(savename,"%s_edited.bmp",savename);
    image->save(savename,BUFFER);

    cout << endl << "Imagem salva em "<< savename << endl;
    cout << "Abrindo imagem..." << endl << endl;
    system(savename);
}

void menuTransformations(Imagem *image)
{
    system("cls");

    int option2;

    cout<<"1. Media (passa-baixas)"<<endl;
    cout<<"2. Gaussiano"<<endl;
    cout<<"3. Derivada Primeira"<<endl;
    cout<<"4. Laplace"<<endl;
    cout<<"5. Enter custom kernel"<<endl;
    cout<<"6. Somar editada com original"<<endl;
    cout<<"7. Cancel"<<endl;
    cout<<endl;

    cout<<"enter your option: ";
    cin >> option2;
    while(!cin){
        cin.clear();
        cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        gotoxy(20,wherey()-1); cout<<"______________________"; gotoxy(20,wherey());
        cin >> option2;
    }
    switch(option2)
    {
        case 1:
            image->media();
            WAIT;
            break;
        case 2:
            image->gaussiano();
            WAIT;
            break;
        case 3:
            image->derivadaPrimeira();
            WAIT;
            break;
        case 4:
            image->laplaciano();
            WAIT;
            break;
        case 5:
            int n;

            cout<<"Enter square matrix dimensions: ";
            cin>>n;
            while(!cin){
                cin.clear();
                cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                gotoxy(32,wherey()-1); cout<<" "; gotoxy(32,wherey());
                cin >> n;
            }

            image->customMask(n);
            WAIT;
            break;
        case 7:
            image->soma();
            WAIT;
            break;
    }
}
```

```
int main()
{
        Imagem image; //classe geral da imagem sendo trabalhada

        menuLoad(&image);

    int option;
    do{

            system("cls");
            gotoxy(1,1);

            cout<<"LOADED: "<<image.imagePath<<endl;
            cout<<endl;
            cout<<"1. Load different image"<<endl;
            cout<<"2. Save image & display"<<endl;
            cout<<"3. Transformations"<<endl;
            cout<<"4. EXIT"<<endl;
            cout<<endl;


            cout<<"enter your option: ";
            cin >> option;
            while(!cin){
                cin.clear();
                cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                gotoxy(20,wherey()-1); cout<<"                          "; gotoxy(20,
                     wherey());
                cin >> option;
            }


            gotoxy(1,10);

            switch(option)
            {
                case 1:
                    menuLoad(&image);

                    break;
                case 2:
                    menuSave(&image);
                    break;
                case 3:
                    menuTransformations(&image);
                    break;
            }

    }while(option!=4);
}
```

3.2 classImage.h

```
#include "bib\EasyBMP.h"

#define ORIGINAL 1
#define BUFFER 2

#define MAX_URL 1024


class Imagem
{
    private:

                BMP original;
                BMP buffer;


        int *bufalo;
        /*matriz usada durante as conversões para posterior tratamento
                                    de valores incoerentes (evitar truncagem indevida da
                                         biblioteca)*/


                int truncaValor(int valor);
        void printMask(float *mask,int tam);
        void zeraBufalo();
```

```
    public:
        Imagem(){};
        ~Imagem(){};

        char imagePath[MAX_URL];
        int h, w; //medidas da imagem para acesso mais rápido e prático

        //FUNÇÕES BÁSCIAS
        int load(char path[]);
        int save(char path[], int who);

        void convertToGrayScale();


        //TRANSFORMAÇÕES

        void convolucao_nxn(float *mask, int n);

        void laplaciano();
        void media();
        void gaussiano();
        void derivadaPrimeira();

        void customMask(int n);

                void soma(); //soma imagem modificada como imagem original e salva em
                    buffer

};
```

3.3 classImage.cpp

```
#include <stdlib.h>
#include<iostream>
#include<conio2.h>

#include "classImage.h"

using namespace std;


int Imagem::load(char path[]) {
    if( original.ReadFromFile(path) ) {

            free(bufalo);

            strcpy(imagePath,path);

                    w = original.TellWidth();
                    h = original.TellHeight();

                    //buffer.SetSize(w, h);
                    buffer.ReadFromFile(path);

                    bufalo = (int*) malloc(w*h*sizeof(int));
                    zeraBufalo();

                    return 1;
    }
        else return 0;

}

int Imagem::save(char path[], int who) {
        if( who == BUFFER )
                return buffer.WriteToFile( path );
        else
                return original.WriteToFile( path );
}



void Imagem::zeraBufalo()
{
    for(int y=0; y < h; y++)
        for(int x=0; x < w; x++)
            bufalo[x+w*y] = 0;
}

void Imagem::printMask(float *mask, int tam)
{
```

```
    cout<<"-> Matriz de Convolucao"<<endl<<endl;
    for(int i=0; i<tam; i++){
        for( int k=0; k<tam; k++)
            cout << "\t" << mask[i + k*tam] << " ";
        cout << endl;
    }
    cout<<endl<<endl;
}

int Imagem::truncaValor(int valor)
{
            if( valor < 0 )

                                        return 0;

                        else if( valor > 255 )

                                                        return 255;

                                                else

                                                                return valor;
}



void Imagem::convertToGrayScale()
//função retirada do manual da biblioteca EasyBMP
{
    for( int j=0 ; j < h ; j++){

        for( int i=0 ; i < w ; i++){

            int temp = (int) floor(    0.299*original(i,j)->Red
                                     + 0.587*original(i,j)->Green
                                     + 0.114*original(i,j)->Blue );

            original(i,j)->Red = (ebmpBYTE) temp;
            original(i,j)->Green = (ebmpBYTE) temp;
            original(i,j)->Blue =  (ebmpBYTE) temp;

            buffer(i,j)->Red = (ebmpBYTE) temp;
            buffer(i,j)->Green = (ebmpBYTE) temp;
            buffer(i,j)->Blue =  (ebmpBYTE) temp;

        }

        gotoxy(1,wherey()); cout<<"Converting to GrayScale... "<<j*100/h +1<<"% ";
    }
    cout<<endl<<endl;
}




void Imagem::convolucao_nxn(float *mask, int n)
{


    const int delta = (int)floor(n/2);


    for(int x=delta; x < w-delta; x++){

        for(int y=delta; y < h-delta; y++)

            for( int i = 0-delta;  i  <  n-delta; i++)

                for( int k = 0-delta;  k  <  n-delta; k++)

                    bufalo[x+w*y] = bufalo[x+w*y] + (int)(buffer(x+i,y+k)->Red) * mask [
                        i+delta + n*(k+delta)];


        gotoxy(1,wherey()); cout<<"Operando convolucao... "<<x*100/w +1<<"% ";
    }
    cout<<endl<<endl;

    for(int x=0; x < w; x++)

                        for(int y=0; y < h; y++) {
```

```
                                        //algoritmo pode gerar valores negativos e maiores que
                                            255, que não têm interpretação em níveis de cinza
                                        bufalo[x+w*y] = truncaValor(bufalo[x+w*y]);

                                    buffer(x,y)->Red = (ebmpBYTE) bufalo[x+w*y];
                                        buffer(x,y)->Blue = (ebmpBYTE) bufalo[x+w*y];
                                        buffer(x,y)->Green = (ebmpBYTE) bufalo[x+w*y];
                                }


        zeraBufalo();
}




void Imagem::soma()
{
    for(int x=0; x < w; x++)
        for(int y=0; y < h; y++) {
            buffer(x,y)->Red = truncaValor( buffer(x,y)->Red + original(x,y)->Red );
                        buffer(x,y)->Blue = buffer(x,y)->Red;
                        buffer(x,y)->Green = buffer(x,y)->Red;
        }
}



void Imagem::customMask(int n)
{
    float mask[n][n];

    for(int x=0; x<n; x++)
        for(int y=0; y<n; y++)
            mask[x][y] = 0;

    cout<<"-> Matriz de Convolucao"<<endl<<endl;


    int posx=1, posy=wherey();

    for(int y=0; y<n; y++){

        for(int x=0; x<n; x++){

            gotoxy( posx+(x+2)*8 , posy+y );

            cin >> mask[x][y];
            while(!cin){
                cin.clear(); cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n
                    ');
                gotoxy( posx+(x+2)*8 , posy+y ); cout<<" \a"; gotoxy( posx+(x+2)*8 ,
                     posy+y );
                cin >> mask[x][y];
            }

        }
    }


    convolucao_nxn((float*)mask,n);
}

void Imagem::laplaciano()
{
    float mask[3][3] = { -1, -1, -1,
                         -1,  8, -1,
                         -1, -1, -1 };

    printMask((float*)mask,3);
    convolucao_nxn((float*)mask,3);
}
void Imagem::media()
{
    float mask[3][3] = { (float)1/9, (float)1/9, (float)1/9,
                         (float)1/9, (float)1/9, (float)1/9,
                         (float)1/9, (float)1/9, (float)1/9 };

    printMask((float*)mask,3);
    convolucao_nxn((float*)mask,3);
```

```
}
void Imagem::gaussiano()
{
    float mask[3][3] = { 0.0625, 0.125 ,0.0625,
                         0.125,  0.25,  0.125,
                         0.0625, 0.125 ,0.0625 };

    printMask((float*)mask,3);
    convolucao_nxn((float*)mask,3);
}
void Imagem::derivadaPrimeira()
{
    float mask[3][3] = {  0, -1,  0,
                         -1,  4, -1,
                          0, -1,  0 };

    printMask((float*)mask,3);
    convolucao_nxn((float*)mask,3);
}
```