UNIVERSIDADE FEDERAL DO ESPIRITO SANTO

ESTRUTURA DE DADOS I

ENGENHARIA DA COMPUTAÇÃO

MATHEUS BATISTA SILVA

RELATÓRIO: EXERCÍCIO PROGRAMA 1-(EP1)

SÃO MATEUS - ES

2021

Uma imagem é basicamente uma matriz, com altura (número de linhas) e largura (número de colunas), e cada elemento da matriz é chamado de pixel (picture element), que possui uma “cor”. Na sua forma mais básica podemos representar um pixel como aceso (“branco”), ou apagado (“preto”). Essas imagens são chamadas de binárias e podem ser representadas de forma bem compacta, já que precisamos apenas de 1 bit por pixel. No entanto, muitas vezes, dois níveis são insuficientes para representar uma imagem em “preto e branco”, pois estas costumam possuir vários níveis ou tons de cinza. Uma alternativa é representar uma imagem em tons de cinza utilizando um byte (8 bits) para cada pixel. Assim, podemos representar imagens com até 256 níveis de cinza por pixel. Já uma imagem colorida requer mais informação para cada pixel. A representação mais comum é obtida decompondo uma cor nas componentes red (vermelho), green (verde) e blue (azul) ou rgb. A partir dessas cores podemos representar um grande espectro cromático de cores, por isso elas são chamadas de cores primárias.

Imagens gravadas no formato PPM contêm os valores de cada pixel da imagem, linha por linha. Nesse formato, as cores dos pixels são gravadas como números de 0 a 255.

- **Objetivo**

  No exercício proposto, buscamos implementar funções em um projeto pré estruturado para processamento de imagem usando OpenGL + GLUT na linguagem C. Nesse EP, implementamos um TAD (Tipo Abstrato de Dado) para operar com imagens do tipo PPM. Nosso objetivo final é produzir um algoritmo que trata imagens, com função semelhante ao Photoshop. Implementamos funções que realizam alterações especificas na imagem tratada, os filtros.

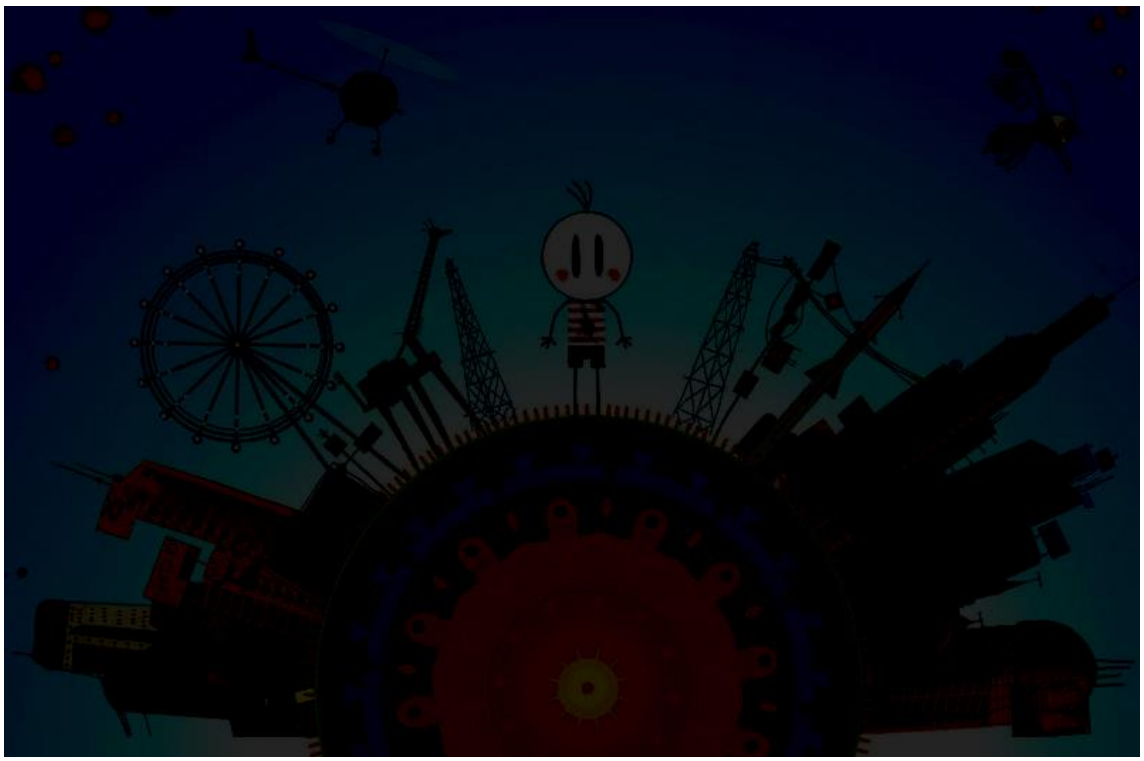
IMAGEM ORIGINAL:

FILTRO DE ESCURECIMENTO

Por meio da função “*escurecerImagem*”, esse filtro reduz de forma proporcional o brilho de cada pixel da imagem. O fator de escurecimento é informado pelo usuário, sendo reduzido um certo valor de cada banda de cor de cada pixel, considerando o intervalo de 0 a 255.

```
22  void escurecerImagem(Imagem *img){
23      int v;
24      printf("Digite o fator de escurecimento: ");
25      scanf("%d", &v);
26      /* Cada canal de cor (RGB) de cada pixel é reduzido 'v' do valor.
27       * Note que devemos garantir que o valor esteja entre 0 e 255.
28       * Como estamos subtraindo, verificamos apenas se o valor é >= 0
29       * Note também a utilização de expressão ternária e o cast (conversão)
30       * entre os valores Byte (unsigned int) e int. Esse cast evita erros nas
31       * operações matemáticas.
32       */
33      for (int h = 0; h < obtemAltura(img); h++) {
34          for (int w = 0; w < obtemLargura(img); w++) {
35              //Obtém o pixel da posição (h, w) da imagem
36              Pixel pixel = obtemPixel(img, h, w);
37              //Modifica cada canal de cor do pixel
38              pixel.cor[RED]   = (((int)pixel.cor[RED]   - v) >= 0 ? ((int)pixel.cor[RED] - v) : 0);
39              pixel.cor[GREEN] = (((int)pixel.cor[GREEN] - v) >= 0 ? ((int)pixel.cor[GREEN] - v) : 0);
40              pixel.cor[BLUE]  = (((int)pixel.cor[BLUE]  - v) >= 0 ? ((int)pixel.cor[BLUE] - v) : 0);
41              //Grava o novo pixel na posição (h, w) da imagem
42              recolorePixel(img, h, w, pixel);
43          }
44      }
45  }
```

Fator de escurecimento 50.



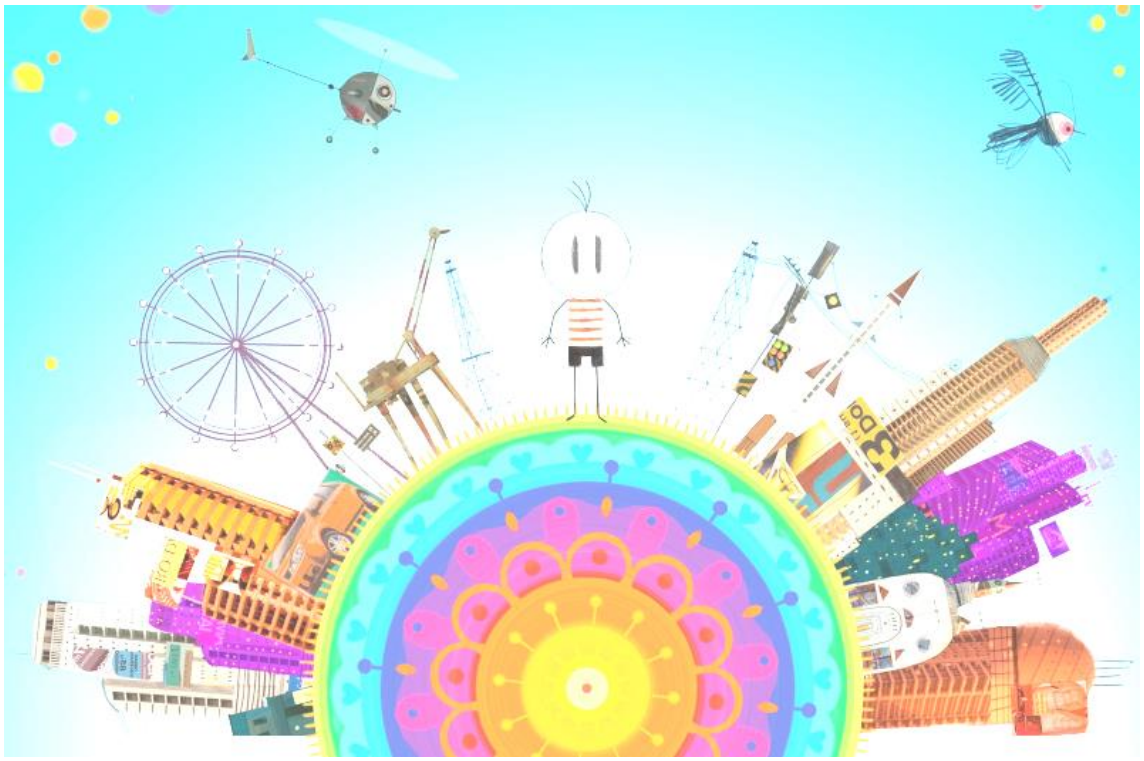
Fator de escurecimento 220.

UFES

FILTRO DE CLAREAMENTO

Por meio da função “*clarearImagem*”, esse filtro aumenta de forma proporcional o brilho de cada pixel da imagem. O fator de clareamento é informado pelo usuário. Abaixo a função implementada e exemplos.

```
51  void clarearImagem(Imagem *img){
52
53      int v;
54      printf("Digite o fator de clareamento: ");
55      scanf("%d", &v);
56
57      for (int h = 0; h < obtemAltura(img); h++) {
58          for (int w = 0; w < obtemLargura(img); w++) {
59              Pixel pixel = obtemPixel(img, h, w);
60              pixel.cor[RED]   = (((int)pixel.cor[RED]   + v) <= 255 ? (pixel.cor[RED]   + v) : 255);
61              pixel.cor[GREEN] = (((int)pixel.cor[GREEN] + v) <= 255 ? (pixel.cor[GREEN] + v) : 255);
62              pixel.cor[BLUE]  = (((int)pixel.cor[BLUE]  + v) <= 255 ? (pixel.cor[BLUE]  + v) : 255);
63
64              recolorePixel(img, h, w, pixel);
65          }
66      }
67
68  }
69
```



Fator de clareamento 80.

Fator de clareamento 210.

FILTRO EM ESCALA DE CINZA

Por meio da função “*escalaDeCinzaImagem*”, esse filtro, para cada pixel, calcula a média dos valores de cada banda de cor e atribuir esse valor a cada componente de cor do pixel, ou seja, $media = (R + G + B)/3$. Cada banda do pixel receberá o fator $media$. Abaixo a função implementada e exemplos.

UFES

```
77  void escalaDeCinzaImagem(Imagem *img){
78
79      //Calculamos a media dos valores R-G-B de um Pixel, e atribuimos essa media a cada um desses Pixels
80      int media;
81      for (int i = 0 ; i <obtemAltura(img) ; i ++) {
82          for (int j = 0 ; j <obtemLargura(img) ; j ++) {
83
84              Pixel pixel = obtemPixel(img,i,j);
85
86              media=((int)(pixel.cor[RED]) + (pixel.cor[GREEN]) + (pixel.cor[BLUE]))/3;
87              pixel.cor[RED]   = media;
88              pixel.cor[GREEN] = media;
89              pixel.cor[BLUE]  = media;
90
91              recolorePixel(img, i, j, pixel);
92
93          }
94      }
95
96  }
```

Escala de Cinza.

UFES

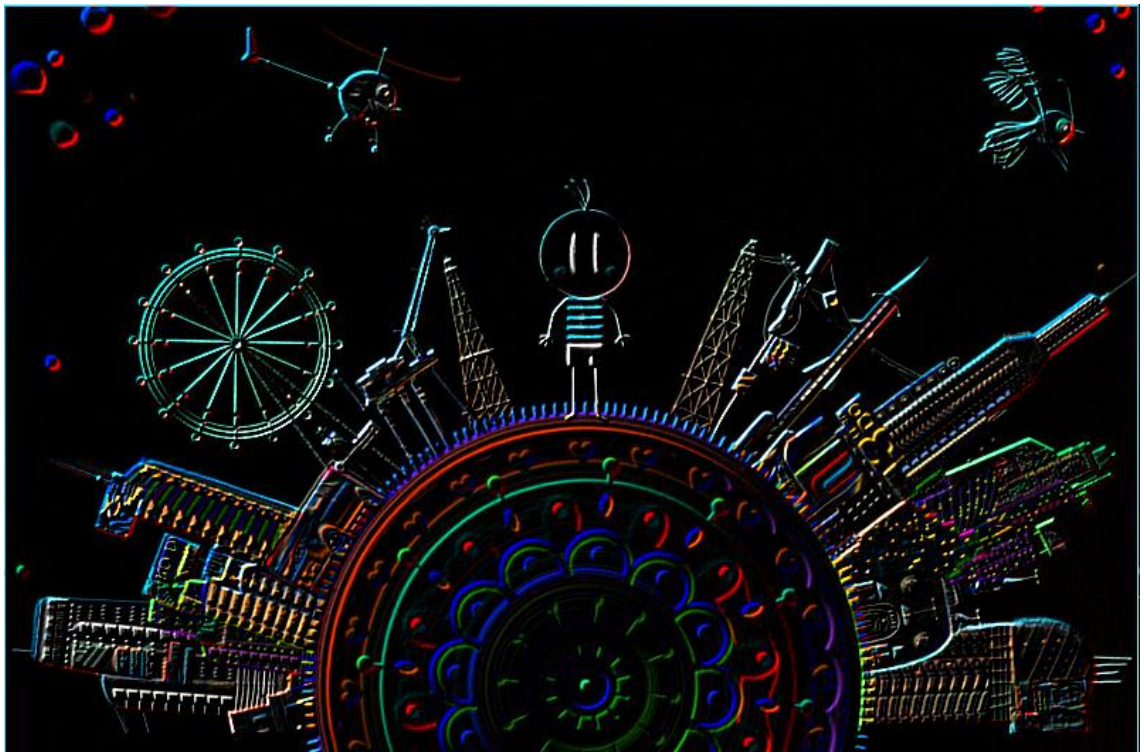
FILTRO DE SOBEL

Por meio da função “*filtroSobel*”, esse filtro pode ser usado para detecção de bordas em imagens digitais e consiste na aplicação de duas matrizes (chamadas de matrizes de convolução), que detectam os contornos na vertical e na horizontal; as matrizes são chamadas de máscara ou Kernel. Nesse EP, utilizamos as seguintes matrizes:

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \text{ e } G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Abaixo a função implementada e exemplos.

```
void filtroSobel(Imagem *img){

    Imagem *copia=copiaImagem(img);
    int A = obtemAltura(img);
    int L = obtemLargura(img);
    int i, j;
    //Percorre cada pixel da matriz, dado um determinado indice, percorremos os elementos vizinhos
    for (i=1;i<A-1;i++){
        for(j=1 ;j<L-1;j++){
            Pixel pixel  = obtemPixel(img,i,j);
            Pixel pix1 = obtemPixel(copia,i-1,j-1);
            Pixel pix2 = obtemPixel(copia,i-1,j);
            Pixel pix3 = obtemPixel(copia,i-1,j+1);
            Pixel pix4 = obtemPixel(copia,i,j-1);
            Pixel pix5 = obtemPixel(img,i,j);
            Pixel pix6 = obtemPixel(copia,i,j+1);
            Pixel pix7 = obtemPixel(copia,i+1,j-1);
            Pixel pix8 = obtemPixel(copia,i+1,j);
            Pixel pix9 = obtemPixel(copia,i+1,j+1);

            int gxred=   (((int)pix1.cor[RED])   + (2 * ((int)pix4.cor[RED]))   + ((int)pix7.cor[RED])   - ((int)pix3.cor[RED])    - (2 * ((int)pix6.cor[RED]))   - ((int)pix9.cor[RED]));
            int gxgreen= (((int)pix1.cor[GREEN]) + (2 * ((int)pix4.cor[GREEN])) + ((int)pix7.cor[GREEN]) - ((int)pix3.cor[GREEN])  - (2 * ((int)pix6.cor[GREEN])) - ((int)pix9.cor[GREEN]));
            int gxblue=  (((int)pix1.cor[BLUE])  + 2 * (((int)pix4.cor[BLUE]))  + ((int)pix7.cor[BLUE])  - ((int)pix3.cor[BLUE])   - (2 * ((int)pix6.cor[BLUE]))  - ((int)pix9.cor[BLUE]));

            int gyred=   (((int)pix1.cor[RED])   + (2 * ((int)pix2.cor[RED]))   + ((int)pix3.cor[RED])   - ((int)pix7.cor[RED])    - (2 * ((int)pix8.cor[RED]))   - ((int)pix9.cor[RED]));
            int gygreen= (((int)pix1.cor[GREEN]) + (2 * ((int)pix2.cor[GREEN])) + ((int)pix3.cor[GREEN]) - ((int)pix7.cor[GREEN]) - (2 * ((int)pix8.cor[GREEN])) - ((int)pix9.cor[GREEN]));
            int gyblue=  (((int)pix1.cor[BLUE])  + (2 * ((int)pix2.cor[BLUE]))  + ((int)pix3.cor[BLUE])  - ((int)pix7.cor[BLUE])  - (2 * ((int)pix8.cor[BLUE]))  - ((int)pix9.cor[BLUE]));

            int pixelred   =   ((gyred)+(gxred))/2 ;
            int pixelgreen =   ((gygreen)+(gxgreen))/2;
            int pixelblue  =   ((gyblue)+(gxblue))/2;

            pixelred=   ((pixelred)   >=0   ? (pixelred)   :0);
            pixelgreen= ((pixelgreen) >=0   ? (pixelgreen) :0);
            pixelblue=  ((pixelblue)  >=0   ? (pixelblue)  :0);

            pixelblue=  ((pixelblue)   <= 255 ?  (pixelblue ) : 255);
            pixelred=   ((pixelred)    <=255  ?  (pixelred)   : 255);
            pixelgreen=  ((pixelgreen) <=255  ?  (pixelgreen) : 255);

            pixel.cor[RED]   = pixelred;
            pixel.cor[GREEN] = pixelgreen;
            pixel.cor[BLUE]  = pixelblue ;

            recolorePixel(img, i, j, pixel);
        }

    }
    liberaImagem(copia);
```



Filtro Sobel..

FILTRO DE DETECÇÃO DE BORDAS DE LAPLACE

Por meio da função “*deteccaoBordasLaplace*”, a implementação do filtro de detecção de bordas de Laplace segue a mesma ideia do filtro de Sobel, mas agora existe apenas uma matriz de convolução que deve ser aplicada em cada banda de cor de cada pixel. Nesse EP, utilizamos a seguinte matriz:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Abaixo a função implementada e exemplos.

UFES

```
void deteccaoBordasLaplace(Imagem *img) {

    Imagem *copia=copiaImagem(img);
    int A = obtemAltura(img);
    int L = obtemLargura(img);
    int i, j;
    //Percorre cada pixel da matriz, dado um determinado indice, percorre os elementos vizinhos, e aplica o efeito, de forma analoga ao filtro Sobel
    for (i=1;i<A-1;i++){
        for(j=1 ;j<L-1;j++){

            Pixel pixel = obtemPixel(img,i,j);
            Pixel pix1 = obtemPixel(copia,i-1,j-1);
            Pixel pix2 = obtemPixel(copia,i-1,j);
            Pixel pix3 = obtemPixel(copia,i-1,j+1);
            Pixel pix4 = obtemPixel(copia,i,j-1);
            Pixel pix5 = obtemPixel(copia,i,j);
            Pixel pix6 = obtemPixel(copia,i,j+1);
            Pixel pix7 = obtemPixel(copia,i+1,j-1);
            Pixel pix8 = obtemPixel(copia,i+1,j);
            Pixel pix9 = obtemPixel(copia,i+1,j+1);

            int pixelred   =   (4 * (((int)pix5.cor[RED]))   -((int)pix2.cor [RED] )   -((int)pix4.cor [RED] )   -((int)pix6.cor [RED] )   - ((int)pix8.cor [RED] ))  ;
            int pixelgreen =   (4 * (((int)pix5.cor[GREEN])) -((int)pix2.cor[GREEN])   -((int)pix4.cor[GREEN])   -((int)pix6.cor[GREEN])   - ((int)pix8.cor[GREEN]))  ;
            int pixelblue  =   (4 * (((int)pix5.cor[BLUE]))  -((int)pix2.cor [BLUE])   -((int)pix4.cor [BLUE])   -((int)pix6.cor [BLUE])   - ((int)pix8.cor [BLUE]))  ;

            pixelred=   ((pixelred)   > 0   ? (pixelred)   :0);
            pixelgreen= ((pixelgreen) > 0   ? (pixelgreen) :0);
            pixelblue=  ((pixelblue)  > 0   ? (pixelblue)  :0);

            pixelblue=   ((pixelblue)   < 255  ?  (pixelblue ) : 255);
            pixelred=    ((pixelred)    < 255  ?  (pixelred)   : 255);
            pixelgreen=  ((pixelgreen)  < 255  ?  (pixelgreen) : 255);



            pixel.cor[RED]   = (pixelred)  ;
            pixel.cor[GREEN] = (pixelgreen);
            pixel.cor[BLUE]  = (pixelblue) ;

            recolorePixel(img, i, j, pixel);

        }}
    liberaImagem(copia);
    /* Siga as mesmas dicas do filtro de Sobel */
}
```

Detecção de Bordas de Laplace.

UFES

MEU FILTRO

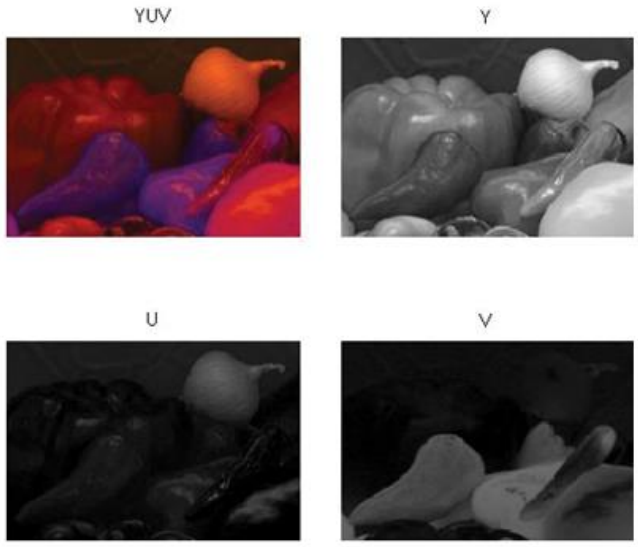
Por meio da função “*meuFiltro*”, implementamos nessa função criativa um “mix” dos conceitos de matriz de convolução e escala de cinza associados a um modelo matemático que é baseado no sistema YIQ. Neste modelo, o componente Y corresponde à luminância e as componentes I (matiz) e Q (saturação) codificam as informações de crominância. O sistema YIQ é utilizado para transmissão de sinal de televisão a cores. O modelo matemático usa a equação mostrada para a produção de componentes YIQ a partir da imagem RGB.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}596 & -0{,}0274 & -0{,}322 \\ 0{,}211 & -0{,}523 & 0{,}311 \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} Y \\ eu \\ Q \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}596 & -\,0{,}0274 & -\,0{,}322 \\ 0{,}211 & -\,0{,}523 & 0{,}311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad E_7$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0{,}986 & 0{,}621 \\ 1 & -\,0{,}0272 & -\,0{,}649 \\ 1 & -\,1.106 & 1{,}703 \end{bmatrix} \begin{bmatrix} Y \\ eu \\ Q \end{bmatrix} \qquad E8$$

```
RGB = imread('onion.png');
R = RGB(:,:,1);
G = RGB(:,:,2);
B = RGB(:,:,3);
Y =  0.299   * R + 0.587   * G + 0.114   * B;
I = -0.14713 * R - 0.28886 * G + 0.436   * B;
Q =  0.615   * R - 0.51499 * G - 0.10001 * B;
YIQ = cat(3,Y,I,Q);
```



Na implementação meuFiltro, nós atribuímos cada elemento da matriz YIQ como fator multiplicativo da matriz de convolução de ordem 3.

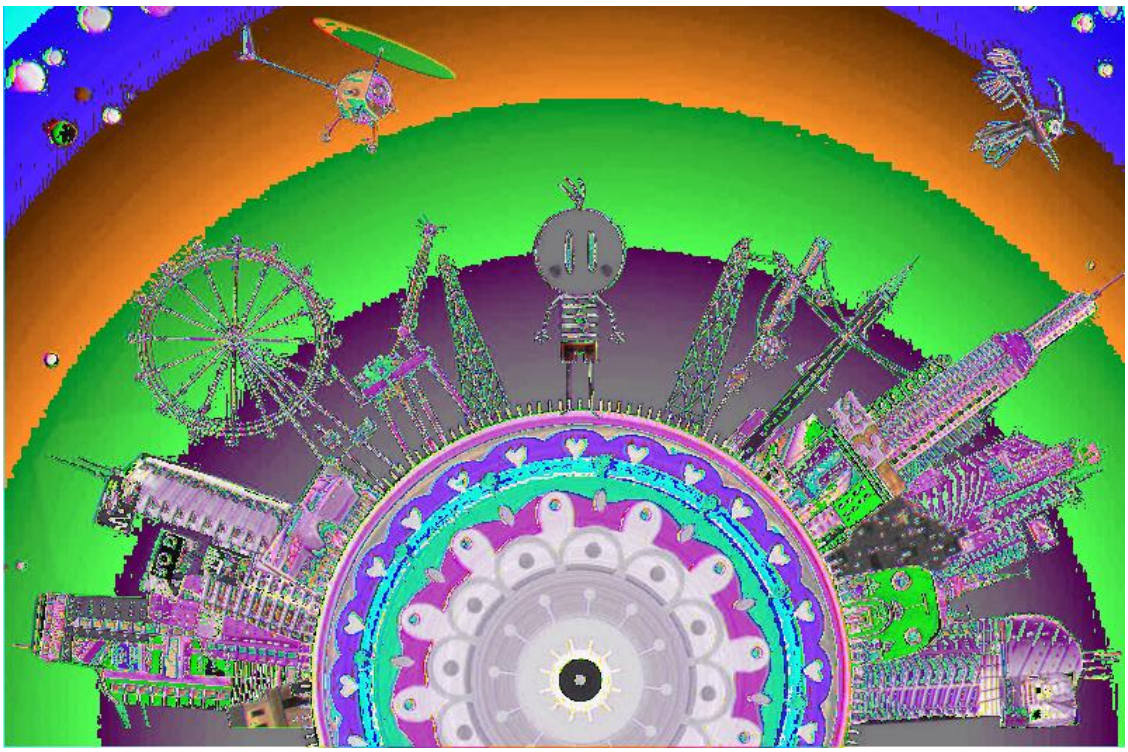
Após perceber um comportamento de destaque de tonalidade dado outro fator de multiplicação, multiplicamos cada banda de cor por um valor pré-definido. No caso, multiplicamos as variáveis: pixelred * 2; pixelgreen * 4; pixelblue * 9.

Por fim dividimos a soma das três bandas de pixel por 2 e armazenamos em uma variável chamada media. Atribuímos a soma das saídas calculadas à saída final do pixel a ser aplicada na imagem.

UFES

```
    for (i=1;i<A-1;i++){
        for(j=1 ;j<L-1;j++){

            Pixel pixel = obtemPixel(img,i,j);
            Pixel pix1 = obtemPixel(copia,i-1,j-1);
            Pixel pix2 = obtemPixel(copia,i-1,j);
            Pixel pix3 = obtemPixel(copia,i-1,j+1);
            Pixel pix4 = obtemPixel(copia,i,j-1);
            Pixel pix5 = obtemPixel(copia,i,j);
            Pixel pix6 = obtemPixel(copia,i,j+1);
            Pixel pix7 = obtemPixel(copia,i+1,j-1);
            Pixel pix8 = obtemPixel(copia,i+1,j);
            Pixel pix9 = obtemPixel(copia,i+1,j+1);


            int pixelred   = ((0.299 * ((int)pix1.cor[RED])) + (0.587 * ((int)pix2.cor[RED]))  + (0.144 * ((int)pix3.cor[RED]))) * 2;
            int pixelgreen = ((0.596 * ((int)pix4.cor[RED])) - (0.0247 * ((int)pix5.cor[RED])) - (0.322 * ((int)pix6.cor[RED]))) * 4;
            int pixelblue  = ((0.211 * ((int)pix7.cor[RED])) - (0.523 * ((int)pix8.cor[RED]))  + (0.311 * ((int)pix9.cor[RED]))) * 9;



            pixelred=   ((pixelred)   > 0   ? (pixelred)   :0);
            pixelgreen= ((pixelgreen) > 0   ? (pixelgreen) :0);
            pixelblue=  ((pixelblue)  > 0   ? (pixelblue)  :0);

            pixelblue=   ((pixelblue)   < 255  ?  (pixelblue ) : 255);
            pixelred=    ((pixelred)    < 255  ?  (pixelred)   : 255);
            pixelgreen=  ((pixelgreen)  < 255  ?  (pixelgreen) : 255);


            int media=((int)(pixel.cor[RED]) + (pixel.cor[GREEN]) + (pixel.cor[BLUE]))/2;

            pixel.cor[RED]   = (pixelred) + media;
            pixel.cor[GREEN] = (pixelgreen) + media;
            pixel.cor[BLUE]  = (pixelblue) + media;

            recolorePixel(img, i, j, pixel);

        }
    }
    liberaImagem(copia);

}
```

UFES



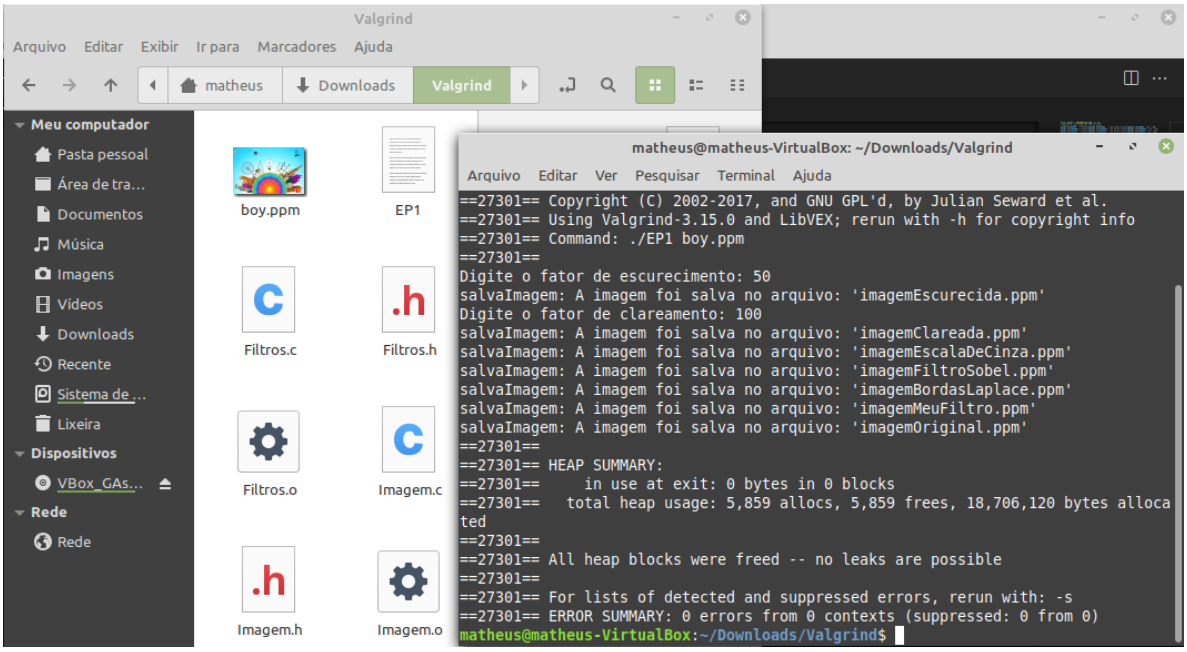
Filtro de autoria própria.

VALGRIND

O teste de vazamento de memória com o valgrind foi positivo com 0 erros.

UFES

```
matheus@matheus-VirtualBox: ~/Downloads/Valgrind
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
==27301== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27301== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==27301== Command: ./EP1 boy.ppm
==27301==
Digite o fator de escurecimento: 50
salvaImagem: A imagem foi salva no arquivo: 'imagemEscurecida.ppm'
Digite o fator de clareamento: 100
salvaImagem: A imagem foi salva no arquivo: 'imagemClareada.ppm'
salvaImagem: A imagem foi salva no arquivo: 'imagemEscalaDeCinza.ppm'
salvaImagem: A imagem foi salva no arquivo: 'imagemFiltroSobel.ppm'
salvaImagem: A imagem foi salva no arquivo: 'imagemBordasLaplace.ppm'
salvaImagem: A imagem foi salva no arquivo: 'imagemMeuFiltro.ppm'
salvaImagem: A imagem foi salva no arquivo: 'imagemOriginal.ppm'
==27301==
==27301== HEAP SUMMARY:
==27301==     in use at exit: 0 bytes in 0 blocks
==27301==   total heap usage: 5,859 allocs, 5,859 frees, 18,706,120 bytes alloca
ted
==27301==
==27301== All heap blocks were freed -- no leaks are possible
==27301==
==27301== For lists of detected and suppressed errors, rerun with: -s
==27301== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
matheus@matheus-VirtualBox:~/Downloads/Valgrind$
```



UFES

- REFERÊNCIAS BIBLIOGRÁFICAS

ROMÃO, Oberlan. Exercício Programa 1; FotoXop. EP1.pdf.

Mahmut sinecen (july 7th 2016). Digital image processing with matlab, applications from engineering with matlab concepts, jan valdman, intechopen, doi: 10.5772/63028. Available from: https://www.intechopen.com/chapters/51312

http://www.decom.ufop.br/guillermo/bcc326/slides/processamento%20de%20imagens%20-%20sistema%20de%20cores.pdf