

TRABALHO DA 1ª UNIDADE – COMPUTAÇÃO GRÁFICA.

Relatório do trabalho de implementação dos métodos de manipulação de imagens.

Marcos André Azevedo de Assis.

Departamento de Computação – Universidade do Estado do Rio Grande do Norte (UERN)
Av. Dr. João Medeiros Filho, nº 3419 – Natal – RN – Brasil.

marcosassis@alu.uern.br, marcosandreaezevedo78@gmail.com

- Os códigos dos métodos para manipulação de imagens podem ser verificados em meu repositório no GitHub: <https://github.com/MarcosAndre5/ComputacaoGrafica>

1. Monocromático:

Uma imagem monocromática pode ser descrita matematicamente por uma função $f(x,y)$ da intensidade luminosa, sendo seu valor, em qualquer ponto de coordenadas espaciais (x,y) , proporcional ao brilho (ou nível de cinza) da imagem naquele ponto.

```
pixel **RGB = (pixel**) malloc (altura * sizeof(pixel*));

for(i = 0; i < altura; i++)
    RGB[i] = (pixel*) malloc (largura * sizeof(pixel));

for(i = 0; i < altura; i++)
    for(j = 0; j < largura; j++){
        fscanf(imagem, "%d", &RGB[i][j].R);
        fscanf(imagem, "%d", &RGB[i][j].G);
        fscanf(imagem, "%d", &RGB[i][j].B);
    }

fprintf(novaImagem, "P2\n%d %d\n%d\n", altura, largura, tamanhoEscala);

for(i = 0; i < altura; i++, fprintf(novaImagem, "\n"))
    for(j = 0; j < largura; j++)
        fprintf(novaImagem, "%d ", (RGB[i][j].R + RGB[i][j].G + RGB[i][j].B) / 3);

printf("Nova Imagem gerada com Sucesso!\n");
fclose(imagem);
fclose(novaImagem);
return 0;
}
```

Código completo pode ser encontrado [aqui](#).

Resultado:



2. Negativo:

O método Negativo é usado para inverter as cores de uma imagem. É muito útil em situações onde a imagem original é escura. Ora com este processamento, os objetos brilhantes mas fracos, passarão a aparecer como objetos escuros contra um fundo claro, tornando, por isso, a visualização mais intuitiva. Um exemplo bastante comum desta técnica são as imagens médicas.

```
pixel **RGB = (pixel**) malloc (altura * sizeof(pixel*));

for(i = 0; i < altura; i++)
    RGB[i] = (pixel*) malloc (largura * sizeof(pixel));

for(i = 0; i < altura; i++)
    for(j = 0; j < largura; j++){
        fscanf(imagem, "%d", &RGB[i][j].R);
        fscanf(imagem, "%d", &RGB[i][j].G);
        fscanf(imagem, "%d", &RGB[i][j].B);
    }

fprintf(novaImagem, "P3\n%d %d\n%d\n", altura, largura, tamanhoEscala);

for(i = 0; i < altura; i++, fprintf(novaImagem, "\n"))
    for(j = 0; j < largura; j++){
        fprintf(novaImagem, "%d ", 255 - RGB[i][j].R);
        fprintf(novaImagem, "%d ", 255 - RGB[i][j].G);
        fprintf(novaImagem, "%d ", 255 - RGB[i][j].B);
    }

printf("Nova Imagem gerada com Sucesso!\n");
fclose(imagem);
fclose(novaImagem);
return 0;
}
```

Código completo pode ser encontrado [aqui](#).

Resultado:



3. Binarização:

Nesta técnica, se obtém como saída uma imagem com apenas dois níveis de luminância: preto e branco. Aqui estou dividindo os pixels, aqueles que possuem tonalidade menor ou igual à 126 será atualizado para preto (0, 0, 0), e os pixels com tonalidade maior que 126 serão atualizados para branco (255, 255, 255).

```

pixel **RGB = (pixel**) malloc (altura * sizeof(pixel*));

for(i = 0; i < altura; i++)
    RGB[i] = (pixel*) malloc (largura * sizeof(pixel));

for(i = 0; i < altura; i++){
    for(j = 0; j < largura; j++){
        fscanf(imagem, "%d", &RGB[i][j].R);
        fscanf(imagem, "%d", &RGB[i][j].G);
        fscanf(imagem, "%d", &RGB[i][j].B);
    }

    fprintf(novaImagem, "P3\n%d %d\n%d\n", altura, largura, tamanhoEscala);

    for(i = 0; i < altura; i++){
        for(j = 0; j < largura; j++){
            if(RGB[i][j].R <= 126){
                RGB[i][j].R = RGB[i][j].G = RGB[i][j].B = 0;
                fprintf(novaImagem, "\n%d ", RGB[i][j].R);
                fprintf(novaImagem, "%d ", RGB[i][j].G);
                fprintf(novaImagem, "%d ", RGB[i][j].B);
            } else if(RGB[i][j].R > 126){
                RGB[i][j].R = RGB[i][j].G = RGB[i][j].B = 255;
                fprintf(novaImagem, "\n%d ", RGB[i][j].R);
                fprintf(novaImagem, "%d ", RGB[i][j].G);
                fprintf(novaImagem, "%d ", RGB[i][j].B);
            }
        }

        printf("Nova Imagem gerada com Sucesso!\n");
        fclose(imagem);
        fclose(novaImagem);
        return 0;
    }
}

```

Código completo pode ser encontrado [aqui](#).

Resultado:



4. Alargamento de Contraste:

É uma técnica que consiste no aumento da escala dinâmica dos níveis de cinza na imagem processada.

```

maior = menor = CINZA[0][0].tomCinza;

for(i = 0; i < altura; i++)
    for(j = 0; j < largura; j++)
        if(CINZA[i][j].tomCinza > maior)
            maior = CINZA[i][j].tomCinza;
        else if(CINZA[i][j].tomCinza < menor)
            menor = CINZA[i][j].tomCinza;

printf("Escala Máxima da Imagem: %d | Escala Mínima da Imagem: %d\n", maior, menor);
printf("Id Máximo: %d | Id Mínimo: %d\n", idMax, idMin);

if((maior - menor) <= 0){
    printf("\nNão é possível aplicar o modelo nessa Imagem!\n");
    fclose(imagem);
    fclose(novaImagem);
    return 0;
}

fprintf(novaImagem, "P2\n%d %d\n%d\n", altura, largura, tamanhoEscala);

for(i = 0; i < altura; i++, fprintf(novaImagem, "\n"))
    for(j = 0; j < largura; j++){
        cinza = (CINZA[i][j].tomCinza - menor) * (idMax - idMin)/(maior - menor) + idMin;
        fprintf(novaImagem, "%d ", cinza);
    }

printf("Nova Imagem gerada com Sucesso!\n");
fclose(imagem);
fclose(novaImagem);
return 0;
}

```

Código completo pode ser encontrado [aqui](#).

Resultado:



5. Fatiamiento:

Podemos interpretar o processo de fatiamiento como sendo a divisão da escala de cinza da imagem original.

```

pixel **CINZA = (pixel**) malloc (altura * sizeof(pixel*));

for(i = 0; i < altura; i++)
    CINZA[i] = (pixel*) malloc (largura * sizeof(pixel));

for(i = 0; i < altura; i++)
    for(j = 0; j < largura; j++)
        fscanf(imagem, "%d", &CINZA[i][j].tomCinza);

fprintf(novaImagem, "P2\n%d %d\n%d\n", altura, largura, tamanhoEscala);

for(i = 0; i < altura; i++, fprintf(novaImagem, "\n"))
    for(j = 0; j < largura; j++)
        if(cinzaMinimo >= CINZA[i][j].tomCinza && CINZA[i][j].tomCinza <= cinzaMaximo)
            fprintf(novaImagem, "%d ", CINZA[i][j].tomCinza);
        else
            fprintf(novaImagem, "%d ", 0);

printf("Nova Imagem gerada com Sucesso!\n");
fclose(imagem);
fclose(novaImagem);
return 0;
}

```

Código completo pode ser encontrado [aqui](#).

Resultado:



6. Escala:

É uma operação de transformações geométricas de processamento de imagens com objetivo de alterar a posição espacial dos pixels que compõem a imagem.

```

pixel **CINZA = (pixel**) malloc (altura * sizeof(pixel*));

for(i = 0; i < altura; i++)
    CINZA[i] = (pixel*) malloc (largura * sizeof(pixel));

for(i = 0; i < altura; i++)
    for(j = 0; j < largura; j++)
        fscanf(imagem, "%d", &CINZA[i][j].tomCinza);

fprintf(novaImagem, "P2\n%d %d\n%d\n", altura, largura, tamanhoEscala);

for(i = 0; i < altura; i++, fprintf(novaImagem, "\n"))
    for(j = 0; j < largura; j++)
        if((i * Sy < altura) && (j * Sx < largura) && (i * Sy >= 0) && (j * Sx >= 0))
            fprintf(novaImagem, "%d ", CINZA[i * Sy][j * Sx].tomCinza);
        else
            fprintf(novaImagem, "%d ", CINZA[i][j].tomCinza);

printf("Nova Imagem gerada com Sucesso!\n");
fclose(imagem);
fclose(novaImagem);
return 0;
}

```

Código completo pode ser encontrado [aqui](#).

Resultado:



7. Adição:

A adição é usada para fazer adição em cada pixel de uma imagem com os pixels de uma outra imagem, para gerar uma só imagem resultante.

```
for(i = 0; i < alt; i++){
    for(j = 0; j < larg; j++){
        if((RGB[i][j].r + RGB2[i][j].r) > 255){
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
        }else if((RGB[i][j].r + RGB2[i][j].r) < 0){
            aux = 0;
            fprintf(novaImagem, "%d ", aux);
        }else
            fprintf(novaImagem, "%d ", (RGB[i][j].r + RGB2[i][j].r));

        if((RGB[i][j].g + RGB2[i][j].g) > 255){
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
        }else if((RGB[i][j].g + RGB2[i][j].g) < 0){
            aux = 0;
            fprintf(novaImagem, "%d ", aux);
        }else
            fprintf(novaImagem, "%d ", (RGB[i][j].g + RGB2[i][j].g));

        if((RGB[i][j].b + RGB2[i][j].b) > 255){
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
        }else if((RGB[i][j].b + RGB2[i][j].b) < 0){
            aux = 0;
            fprintf(novaImagem, "%d ", aux);
        }else
            fprintf(novaImagem, "%d ", (RGB[i][j].b + RGB2[i][j].b));
    }
    fprintf(novaImagem, "\n");
}
```

Código completo pode ser encontrado [aqui](#).

Resultado:



8. Subtração:

A subtração é usada para subtrair cada pixel de uma imagem pelos pixels de outra, para gerar uma só imagem resultante.

```
for(i = 0; i < alt; i++){
    for(j = 0; j < larg; j++){
        if((RGB[i][j].r - RGB2[i][j].r) > 255){
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
        }else if((RGB[i][j].r - RGB2[i][j].r) < 0){
            aux = 0;
            fprintf(novaImagem, "%d ", aux);
        }else
            fprintf(novaImagem, "%d ", (RGB[i][j].r - RGB2[i][j].r));

        if((RGB[i][j].g - RGB2[i][j].g) > 255){
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
        }else if((RGB[i][j].g - RGB2[i][j].g) < 0){
            aux = 0;
            fprintf(novaImagem, "%d ", aux);
        }else
            fprintf(novaImagem, "%d ", (RGB[i][j].g - RGB2[i][j].g));

        if((RGB[i][j].b - RGB2[i][j].b) > 255){
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
        }else if((RGB[i][j].b - RGB2[i][j].b) < 0){
            aux = 0;
            fprintf(novaImagem, "%d ", aux);
        }else
            fprintf(novaImagem, "%d ", (RGB[i][j].b - RGB2[i][j].b));
    }
    fprintf(novaImagem, "\n");
}
```

Código completo pode ser encontrado [aqui](#).

Resultado:



9. Multiplicação:

A multiplicação é usada para multiplicar cada pixel de uma imagem pelos pixels de outra, para gerar uma só imagem resultante.

```

for(i = 0; i < alt; i++){
    for(j = 0; j < larg; j++){
        if((RGB[i][j].r * RGB2[i][j].r) > 255){
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
        }else if((RGB[i][j].r * RGB2[i][j].r) < 0){
            aux = 0;
            fprintf(novaImagem, "%d ", aux);
        }else
            fprintf(novaImagem, "%d ", (RGB[i][j].r * RGB2[i][j].r));

        if((RGB[i][j].g * RGB2[i][j].g) > 255){
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
        }else if((RGB[i][j].g * RGB2[i][j].g) < 0){
            aux = 0;
            fprintf(novaImagem, "%d ", aux);
        }else
            fprintf(novaImagem, "%d ", (RGB[i][j].g * RGB2[i][j].g));

        if((RGB[i][j].b * RGB2[i][j].b) > 255){
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
        }else if((RGB[i][j].b * RGB2[i][j].b) < 0){
            aux = 0;
            fprintf(novaImagem, "%d ", aux);
        }else
            fprintf(novaImagem, "%d ", (RGB[i][j].b * RGB2[i][j].b));
    }
    fprintf(novaImagem, "\n");
}

```

Código completo pode ser encontrado [aqui](#).

Resultado:



10. Divisão:

A divisão é usada para dividir cada pixel de uma imagem pelos pixels de outra, para gerar uma só imagem resultante.


```

for(i = 0; i < alt; i++){
    for(j = 0; j < larg; j++){
        if((RGB[i][j].r / RGB2[i][j].r) > 255){
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
        }else if((RGB[i][j].r / RGB2[i][j].r) < 0){
            aux = 0;
            fprintf(novaImagem, "%d ", aux);
        }else
            fprintf(novaImagem, "%d ", (RGB[i][j].r / RGB2[i][j].r));

        if((RGB[i][j].g / RGB2[i][j].g) > 255){
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
        }else if((RGB[i][j].g / RGB2[i][j].g) < 0){
            aux = 0;
            fprintf(novaImagem, "%d ", aux);
        }else
            fprintf(novaImagem, "%d ", (RGB[i][j].g / RGB2[i][j].g));

        if((RGB[i][j].b / RGB2[i][j].b) > 255){
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
        }else if((RGB[i][j].b / RGB2[i][j].b) < 0){
            aux = 0;
            fprintf(novaImagem, "%d ", aux);
        }else
            fprintf(novaImagem, "%d ", (RGB[i][j].b / RGB2[i][j].b));
    }
    fprintf(novaImagem, "\n");
}

```

Código completo pode ser encontrado [aqui](#).

Resultado:

11. E Lógico:

No E Lógico é feito uma comparação pixel a pixel das duas imagens, quando os pixels de ambas as imagens e mesma posição tem tonalidades iguais e tonalidade é gravada na nova imagem, já quando as tonalidades são diferentes a tonalidade na imagem nova é gravada como branco.

```

for(i = 0; i < alt; i++){
    for(j = 0; j < larg; j++){
        if((RGB[i][j].r == RGB2[i][j].r) && (RGB[i][j].g == RGB2[i][j].g) && (RGB[i][j].b == RGB2[i][j].b)){
            fprintf(novaImagem, "%d ", RGB[i][j].r);
            fprintf(novaImagem, "%d ", RGB[i][j].g);
            fprintf(novaImagem, "%d ", RGB[i][j].b);
        }else{
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
            fprintf(novaImagem, "%d ", aux);
            fprintf(novaImagem, "%d ", aux);
        }
    }
    fprintf(novaImagem, "\n");
}

```

Código completo pode ser encontrado [aqui](#).

Resultado:



12. OU Lógico:

```
for(i = 0; i < alt; i++){
    for(j = 0; j < larg; j++){
        if((RGB[i][j].r == RGB2[i][j].r) || (RGB[i][j].g == RGB2[i][j].g) || (RGB[i][j].b == RGB2[i][j].b)){
            fprintf(novaImagem, "%d ", RGB[i][j].r);
            fprintf(novaImagem, "%d ", RGB[i][j].g);
            fprintf(novaImagem, "%d ", RGB[i][j].b);
        }else{
            aux = 255;
            fprintf(novaImagem, "%d ", aux);
            fprintf(novaImagem, "%d ", aux);
            fprintf(novaImagem, "%d ", aux);
        }
    }
    fprintf(novaImagem, "\n");
}
```

Código completo pode ser encontrado [aqui](#).

Resultado:



13. NÃO Lógico:

No NÃO Lógico é feito uma inversão pixel a pixel de uma imagem. Aqui eu usei uma imagem binária onde a imagem resultante obteve branco onde antes era preto e preto onde antes era branco.

```
for(i = 0; i < alt; i++)
    for(j = 0; j < larg; j++){
        if(RGB[i][j].r == 255){
            RGB[i][j].r = 0;
            RGB[i][j].g = 0;
            RGB[i][j].b = 0;
            fprintf(novaImagem, "\n%d ", RGB[i][j].r);
            fprintf(novaImagem, "%d ", RGB[i][j].g);
            fprintf(novaImagem, "%d ", RGB[i][j].b);
        }else if(RGB[i][j].r == 0){
            RGB[i][j].r = 255;
            RGB[i][j].g = 255;
            RGB[i][j].b = 255;
            fprintf(novaImagem, "\n%d ", RGB[i][j].r);
            fprintf(novaImagem, "%d ", RGB[i][j].g);
            fprintf(novaImagem, "%d ", RGB[i][j].b);
        }
    }
}
```

Código completo pode ser encontrado [aqui](#).

Resultado:

$$\text{NÃO}(\text{img}) = \text{img}$$

Referências:

- <https://www.ogemarques.com/wp-content/uploads/2014/11/pdi99.pdf>