1 Leitura e Gravação de Arquivos de Imagens

Para a leitura e gravação das imagens, eu utilizei as bibliotecas stb_image.h e stb_image_write.h, que estão no domínio público. Ao ler e escrever imagens sem alterá-las, os tamanhos dos arquivos tendem a aumentar. Isso se deve ao fato de JPEG ser um formato de arquivo comprimível, e, no momento, não estou comprimindo as imagens ao salvá-las. A função de escrita possui um parâmetro 'quality' que corresponde à taxa de compressão que deve ser utilizada.

2 Exibição e Operações sobre Imagens

Eu utilizei a biblioteca GTK para construir esse pequeno app. O UI consiste de três janelas: uma com os botões para aplicar as operações, uma para exibir a imagem de entrada, e uma para exibir a imagem de saída. Com certeza a parte mais difícil desse trabalho foi aprender uma nova biblioteca para contrução da interface.

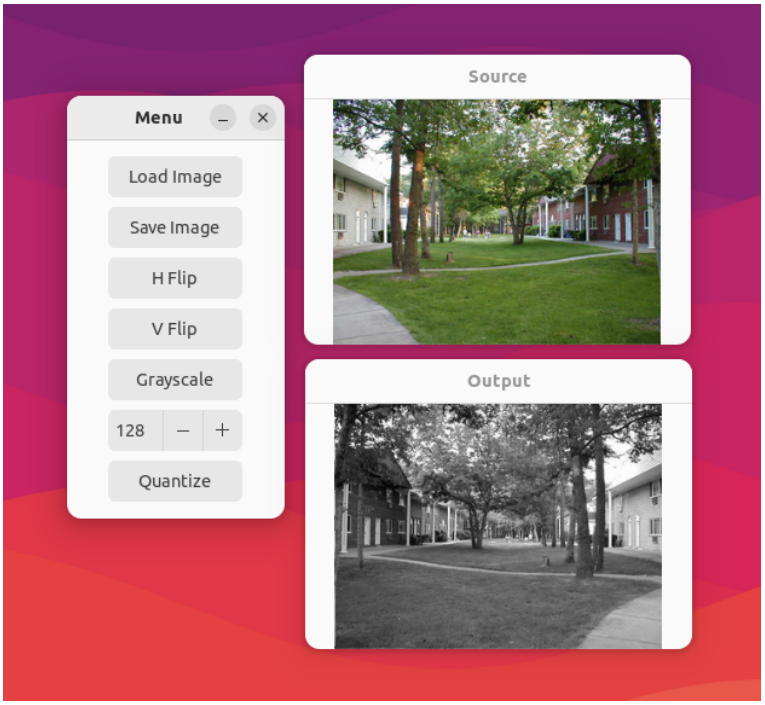

Figura 1: Interface

2.1 Espelhamento Horizontal e Vertical

Não houveram dificuldades para realizar esta seção. Segue o código desenvolvido.

```
#define map(i, j, k, x, n) (i*x*n + j*n + k)

void vflip(unsigned char* data, int x, int y, int n) {
    char *tmp = malloc(x*n);
    for (int i = 0; i < (int) (y/2); i ++) {
        memcpy(tmp, data + i*x*n, x*n);
        memcpy(data + i*x*n, data + (y-i-1)*x*n, x*n);
        memcpy(data + (y-i-1)*x*n, tmp, x*n);
```

```
    }
    free(tmp);
}

void hflip(unsigned char *data, int x, int y, int n) {
    char *tmp = malloc(n);
    for (int j = 0; j < (int) (x/2) ; j++) {
        for (int i = 0; i < y; i++) {
            memcpy(tmp, data + map(i,j,0,x,n), n);
            memcpy(data + map(i,j,0,x,n), data + map(i,(x-j),0,x,n), n);
            memcpy(data + map(i,(x-j),0,x,n), tmp, n);
        }
    }
    free(tmp);
}
```

(a) Horizontal

(b) Original

(c) Vertical

Figura 2: Comparando espelhamentos

2.2 Conversão de Imagem Colorida para Tons de Cinza (Luminância)

Não houveram dificuldades para realizar esta seção. Segue o código desenvolvido.

```
void rgb_to_l(unsigned char* data, int x, int y, int n) {
    int Ri, Gi, Bi;
    unsigned char L;
    for (int i = 0; i < y; i++) {
        for (int j = 0; j < x; j++) {
            Ri = map(i,j,0,x,n);
            Gi = map(i,j,1,x,n);
            Bi = map(i,j,2,x,n);
            L = (unsigned char) (0.299*data[Ri] +
                                 0.587*data[Gi] +
                                 0.114*data[Bi]);
            memset(&data[Ri], L, n);
        }
    }
}
```

(a) Original

(b) Tons de Cinza

Figura 3: Transformação em tons de cinza

2.3 Quantização sobre Imagens em Tons de Cinza

Não houveram dificuldades para realizar esta seção. Segue o código desenvolvido.

```
void l_quantize(unsigned char *data, int x, int y, int n, int q) {
    // Find t1 and t2 (min and max)
    int t1 = data[0];
    int t2 = data[0];
    for (int i = 0; i < x*y*n; i += n) {
        t1 = (data[i] < t1)? data[i] : t1; // min
        t2 = (data[i] > t2)? data[i] : t2; // max
    }
    int int_size = t2 - t1 + 1;
    if (q >= int_size)
        return; // No quantization is necessary

    // Bin and quantize
    float bin_size = (float) int_size / (float) q;
    int L, bin_id;
    float Li, Lj;
    for (int i = 0; i < x*y*n; i += n) {
        bin_id = (data[i] - t1) / bin_size;
        Li = t1 + bin_id * bin_size;
        Lj = t1 + (bin_id+1) * bin_size;
        L = round((double)(Li + Lj)/2);
        memset(&data[i], L, n);
    }
}
```

(a) Original

(b) q = 32

(c) q = 16

Figura 4: Quantização de tons

2.4 Salvamento da Imagem Resultante

É simples salvar o arquivo da imagem resultante.

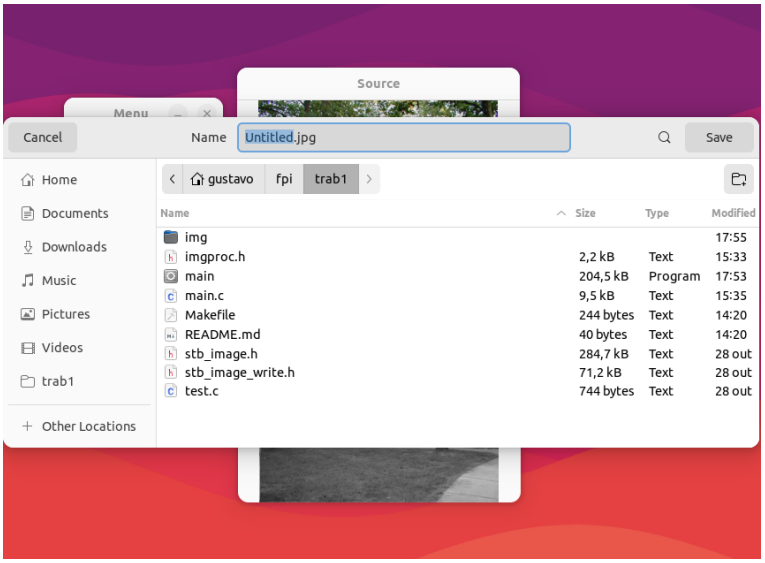

Figura 5: Interface

3 Comentários Finais

Como já mencionei, a maior dificuldade foi criar a interface gráfica. Talvez por minha escolha de biblioteca, GTK, algumas coisas foram mais difíceis do que necessário. Eu consideraria utilizar Qt como alternativa, pois oferece uma solução mais low-code.

A operação que eu mais gostei foi a de quantização. Muito divertido pegar uma foto e tentar chutar o menor número de quantização sem que a foto perca detalhes importantes.