Pontifícia Universidade Católica do Rio Grande do Sul – PUCRS
Escola Politécnica – Curso de Bacharelado em Engenharia de Software
Programação de Software Básico – Turma 31 – Prof. Roland Teodorowitsch – 2023/1

TRABALHO 2: *QUADTREE* PARA COMPRESSÃO DE IMAGENS

(adaptado de uma definição elaborada pelo prof. Marcelo Cohen)

1. Introdução

Ha diversas técnicas que podem ser usadas para a compressão de imagens. As mais conhecidas são os algoritmos implementados no padrão JPEG (*Joint Photographic Experts Group*) ou PNG (*Portable Network Graphics*), por exemplo. Porém, há outras técnicas menos conhecidas: uma delas é baseada no uso de uma árvore para a representação da imagem. Essa técnica, conhecida como representação através de subdivisão por ocupação espacial, utiliza uma estrutura de dados denominada *quadtree*, pois é uma árvore onde cada nodo pode ter zero ou quatro nodos "filhos".

O objetivo deste trabalho é explorar os conceitos de programação C, bem como o uso de ponteiros em uma estrutura de árvore, criando um programa capaz de ler uma imagem qualquer e gerar uma *quadtree* que corresponde à representação dessa imagem. Para visualizar o resultado, é fornecido um código que desenha a árvore.

2. Funcionamento

Será fornecido um *template* de projeto que contém a estrutura de uma aplicação para leitura e visualização da imagem e da *quadtree* correspondente. Para leitura das imagens, este projeto utiliza uma biblioteca chamada SOIL (*Simple OpenGL Image Library*), desenvolvida por Jonathan Dummer.

O programa principal do projeto, ao ser iniciado, executa os seguintes passos:

1. Leitura de uma imagem colorida, onde cada *pixel* (ponto da imagem) é representado em RGB (*Red, Green, Blue*), ou seja, para cada componente das cores vermelho, verde e azul armazena-se a respectiva intensidade utilizando um `unsigned char`.

2. Obtenção do menor erro desejado, o que deve ser fornecido pelo usuário. Esse valor varia de imagem para a imagem, e representa, em linhas, gerais, o desvio padrão das intensidades dos *pixels* de cada região.

3. Geração da *quadtree* (que deve ser implementado) e chamada das funções para visualização e para salvamento da *quadtree* em arquivo (o que já é fornecido).

Maiores detalhes sobre o projeto, suas bibliotecas e seu código-fonte são fornecidos na Seção 4.

O programa de exemplo recebe o nome da imagem a ser carregada pela linha de comando, como o primeiro parâmetro. A imagem original é então exibida em uma janela, e pode-se usar as seguintes teclas:

- ESC: libera memória e termina o programa;
- =: aumenta em uma unidade o nível de erro atual, recriando a árvore;
- -: reduz em uma unidade o nível de erro atual, recriando a árvore;
- b: liga/desliga o desenho das bordas de cada região;
- r: recria e desenha a árvore, sem alterar o nível de erro;
- w: grava a árvore no disco, no formato de entrada do Graphviz (.dot).

Será, no entanto, necessário gerar a *quadtree* para poder visualizar as bordas de suas regiões nessa ferramenta. A próxima seção descreve os procedimentos necessários para geração de uma *quadtree*.

3. Etapas do algoritmo de geração da *quadtree*

O processo de geração da *quadtree* é um algoritmo recursivo: a raiz da árvore representa toda a região da imagem. Se essa região não tem muitos detalhes, o processo se encerra. Caso contrário, são gerados nodos filhos para cada subregião: superior esquerda (NW), superior direita (NE), inferior esquerda (SW) e inferior direita (SE). E o algoritmo é novamente aplicado para cada uma delas. Para entender o processo, veja a sequência de imagens, exibidas na Figura 1, que mostra o resultado ao algoritmo para 1, 2, 3, 4 e 5 níveis na árvore.

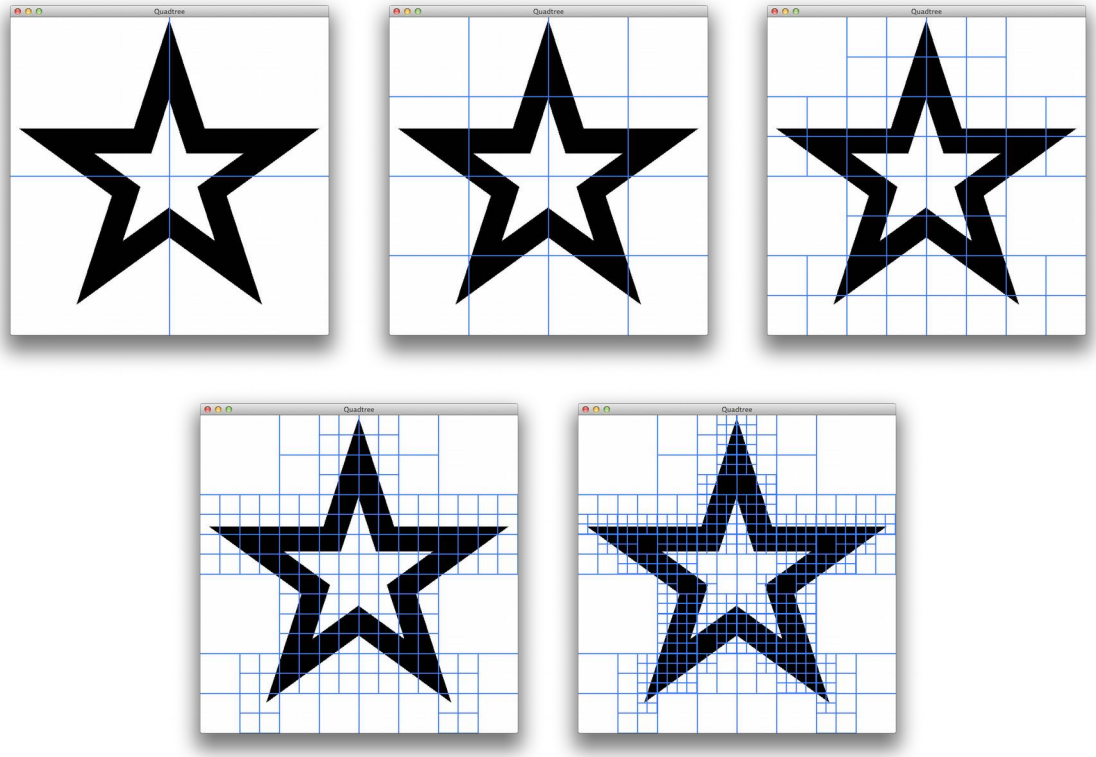

Figura 1 – Aplicação do algoritmo da quadtree para a imagem `star.png`

A última imagem mostra o resultado final, ou seja, ao chegar no nível de erro desejado. Esse nível é especificado através de um valor, que deve ser comparado com o nível de erro da região (ver Seção 3.3): se este último for inferior ou igual ao valor informado, significa que o nível desejado foi atingido e o processo se encerra. Note que regiões com apenas uma cor têm nível de erro igual a zero (Figura 2).

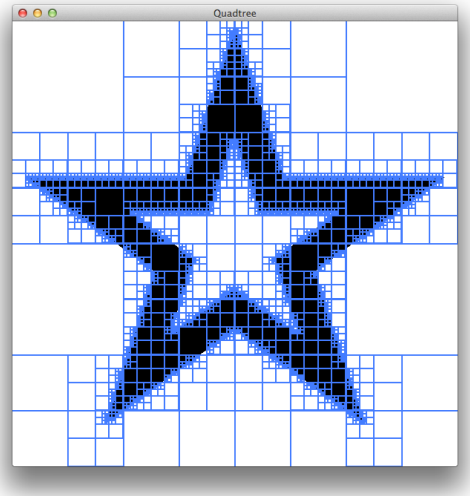

Figura 2 – Resultado da imagem `star.png` até chegar ao nível de erro zero

A Figura 3 apresenta um exemplo de árvore gerada, para uma imagem simples.

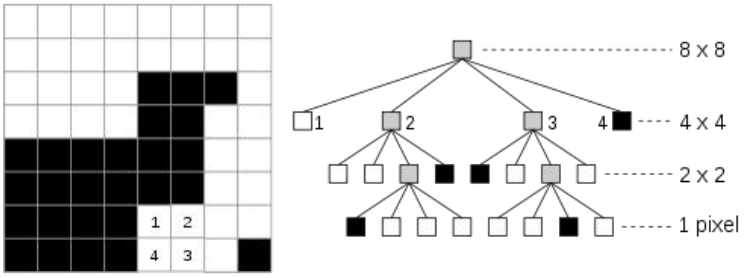

Figura 3 – Exemplo de árvore para uma imagem simples

As próximas seções detalham as etapas do algoritmo em si:

1. Converter a imagem para tons de cinza (seção 3.1), isto é, gerar uma imagem onde cada *pixel* possui uma intensidade (0 - preto, 255 - branco). Essa imagem será utilizada para decidir o momento de encerrar o algoritmo, através do histograma de cada região.

2. Inicialmente só haverá uma única região, abrangendo a imagem inteira, mas à medida que o algoritmo for executado, serão criadas novas regiões para as quais será necessário:

    - calcular a cor média da região (Seção 3.4);
    - calcular o histograma da região em tons de cinza (seção 3.2);
    - Calcular o nível de erro da região, através do histograma (seção 3.3).
    - Se o nível de erro for inferior ao erro mínimo especificado pelo usuário, o processo termina para essa região;
    - Caso contrário, subdividir a região em 4 e repetir o algoritmo, recursivamente.

3.1. Conversão da imagem para tons de cinza

Para converter a imagem para tons de cinza, deve-se alocar uma área de memória (matriz) cujas dimensões sejam iguais às da imagem de entrada. Porém com apenas um *byte* por *pixel*: esse *byte* representará a intensidade luminosa equivalente ao *pixel* correspondente, de acordo com a fórmula:

$$I_{ij} = 0.3\,R_{ij} + 0.59\,G_{ij} + 0.11\,B_{ij}$$

Onde I_{ij} é a intensidade resultante para o *pixel* (i,j) da imagem original. A média ponderada presente nessa fórmula vem do fato que o olho humano tem mais sensibilidade a tons de verde, um pouco menos a tons de vermelho, e consideravelmente menos a tons de azul.

A Figura 4 mostra um exemplo de resultado desse processo aplicado a uma imagem colorida.

Figura 4 – Exemplo de conversão de uma imagem colorida para tons de cinza

3.2. Cálculo do histograma de cada região em tons de cinza

Um histograma permite visualizar a distribuição de tons de uma imagem, dos mais escuros aos mais claros. A Figura 5 mostra um exemplo de histograma considerando uma região do tamanho da imagem.

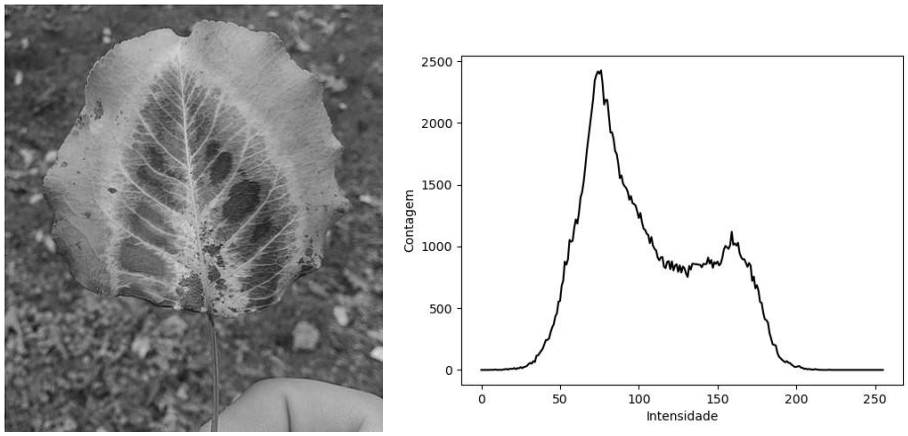

Figura 5 – Exemplo de histograma de uma imagem

Ou seja, o histograma conta a frequência (quantidade de *pixels*) de cada tom de cinza (de 0 a 255) dentro da região. Ele pode ser representado simplesmente por um *array* de inteiros, onde o índice é o tom de cinza e o valor armazenado é a sua frequência.

3.3. Cálculo do nível de erro da região

Para calcular o erro de uma região, deve ser usado o seguinte algoritmo:

1. A partir do histograma da região, calcular a intensidade média ($\bar{I}$) de uma região. Para tanto, deve-se fazer um somatório de cada entrada do histograma multiplicada por sua frequência. A seguir, divide-se essa soma pelo total de *pixels* da região.

2. A seguir pode-se calcular o erro da região através da seguinte fórmula:

$$E = \sqrt{1 frac \frac{1}{wh} \sum_{i=0}^{w-1} \sum_{j=0}^{h-1} (I_{ij} - \bar{I})^2}$$

Onde *w* e *h* são a largura e altura da região, e I_{ij} é a intensidade do pixel na posição (i,j).

3.4. Cálculo da cor média da região

Cada região armazena uma cor média (utilizada no desenho), calculada simplesmente através da média de todas as cores (RGB) dos *pixels* da região.

4. Projeto

A seguir são descritos os aspectos mais relevantes tanto do código-fonte fornecido quando da implementação que deverá ser feita.

4.1. Acessando os *pixels* da imagem

A biblioteca SOIL é responsável pela correta leitura da imagem. O programa principal armazena a imagem em uma *struct* `Img`, conforme mostra a Figura 6.

```
typedef struct {
    int width, height;
    RGBPixel *img;
} Img;
```

Figura 6 – *Struct* `Img`

Ou seja, há a informação de largura e altura, bem como um ponteiro para o vetor com os *pixels* da imagem. Lembre-se que será necessário acessar regiões específicas da imagem, então será preciso converter coordenadas na forma (linha,coluna) para uma posição nesse vetor.

No módulo `quadtree.c`, um trecho da função `geraQuadtree()`, demonstra como acessar as componentes de cor de cada *pixel*, conforme mostra a Figura 7.

```
// pixels é um ponteiro que permite o acesso do vetor img como matriz - veja no código

for(int i=0; i<10; i++)
    printf("%02X %02X %02X\n",pixels[0][i].r,pixels[0][i].g,pixels[0][i].b);
```

Figura 7 – Código para acessar os componentes de cor de cada *pixel*

Este trecho de código exibe, em hexadecimal, as componentes de cor R, G e B dos primeiros 10 *pixels* (a partir do canto superior esquerdo).

4.2. A *struct* `QuadNode`

A estrutura de dados a ser utilizada é a *struct* `QuadNode`, já fornecida. O algoritmo de geração da árvore deve estar presente no módulo `quadtree.c` (também já fornecido), dentro da função `geraQuadTree` (ou ser chamado por ela).

A *struct* `Quad` representa um nodo da *quadtree*, como mostra a Figura 8.

```
enum { CHEIO, PARCIAL };

struct Quad {
    unsigned int id;
    float x, y;              // canto superior esquerdo da região
    float width, height;     // largura e altura da região
    int status;              // CHEIO ou PARCIAL
    unsigned char color[3];  // cor média da região
    struct Quad *NW;         // ponteiros para os filhos, se houver
    struct Quad *NE;
    struct Quad *SW;
    struct Quad *SE;
};

typedef struct Quad QuadNode;
```

Figura 8 – *Struct* `Quad`

A *struct* `Quad` não deve ser alterada, pois é usada dessa forma para desenhar a *quadtree*. Também é declarado um `typedef` para a *struct* `Quad` com o nome `QuadNode`, para facilitar seu uso. As alterações devem ser feitas apenas no módulo `quadtree`, criando funções adicionais, etc. A função `geraQuadtree()` deve ser complementada, incluindo o código que gera os nodos da árvore. Preferencialmente, não altere o restante do programa.

Você pode utilizar a função `newNode` para gerar um novo nodo na árvore: ela recebe as coordenadas *x* e *y*, bem como largura e altura do nodo, alocando memória e devolvendo um ponteiro para `QuadNode`. Observe que você é responsável pelo encadeamento, isto é, a ligação desse nodo com os demais.

```
QuadNode *newNode(int x, int y, int width, int height);
```

Figura 9 – Protótipo da função `newNode()`

5. Compilação

O código-fonte básico do projeto (arquivo `quadtree-base.zip`) está disponível no Moodle da disciplina.

Este *zip* contém o projeto para a implementação do trabalho. Esse código já realiza a leitura de uma imagem qualquer de 24 *bits*. O projeto pode ser compilado no Windows, Linux ou macOS, seguindo as instruções abaixo.

Para a compilação no Linux, é necessário ter instalado os pacotes de desenvolvimento da biblioteca

OpenGL. Para Ubuntu, Mint, Debian e derivados, instale com:

```
sudo apt-get install freeglut3-dev
```

Para a compilação no Windows ou no macOS, não é necessário instalar mais nada – o compilador já vem com as bibliotecas necessárias.

5.1. Visual Studio Code

Se você estiver utilizando o Visual Studio Code, basta descompactar o zip e abrir a pasta.

Para compilar: use Ctrl+Shift+B (⌘+Shift+B no macOS).

Para executar, use F5 para usar o *debugger* ou Ctrl+F5 para executar sem o *debugger*.

5.2. Outros ambientes ou terminal

Caso esteja usando outro ambiente de desenvolvimento, há um `Makefile` para Linux e macOS, e outro para Windows (`Makefile.mk`).

Dessa forma, para compilar no Linux ou macOS, basta digitar:

```
make
```

Se estiver utilizando o Windows, o comando é similar:

```
mingw32-make -f Makefile.mk
```

Alternativamente, você também pode utilizar o `CMake` (*Cross Platform Make*) para compilar pelo terminal. Para tanto, crie um diretório `build` embaixo do diretório do projeto e faça:

```
cd build
cmake ..
make -j  # ou mingw32-make -j no Windows
```

6. Avaliação

Os seguintes critérios serão utilizados na avaliação deste trabalho:

- Os trabalhos deverão ser armazenados em um repositório *git* (preferencialmente GitHub).
- O repositório deve ser mantido privado durante toda a execução do trabalho, para evitar problemas de plágio.

- Compartilhe o repositório com o professor (usuário *github*: rolandteodorowitsch).
- Todos os integrantes do grupo devem ter contribuições (*commits*) no repositório.
- Caso o programa apresente algum problema na execução, a seguinte escala de pontos será considerada:
  - Conversão da imagem para tons de cinza: 1 ponto;
  - Cálculo da cor média de cada região: 1 ponto;
  - Cálculo do histograma de cada região: 1,5 pontos;
  - Cálculo do nível de erro de cada região: 1,5 pontos;
  - Geração da *quadtree* (subdivisão e encadeamento): 5 pontos;
  - Integrante do grupo sem *commits*: -1 ponto.
- Os trabalhos são em duplas ou individuais. A pasta do projeto deve ser compactada em um arquivo .zip e submetida pelo Moodle até a data e hora especificadas. Não envie .rar, .7z, .tar.gz, ou qualquer outro formato esotérico – apenas .zip.
- O código deve estar indentado corretamente (qualquer editor decente faz isso automaticamente).
- A cópia parcial ou completa do trabalho terá como consequência a atribuição de nota ZERO ao trabalho dos alunos envolvidos.
- A cópia de código ou algoritmos existentes da Internet também não é permitida. Se alguma ideia encontrada na rede for utilizada na implementação, sua descrição e referência deve constar como um comentário no código, ou em um arquivo `README.txt`.