Trabalho Prático 2

Compressão de Imagens por *Quadtree*

10/2020

1 Introdução

Ha diversas técnicas que podem ser usadas para a compressão de imagens. As mais conhecidas são os algoritmos implementados no padrão JPEG ou PNG, 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 "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* correspondente. Para visualizar o resultado, é fornecido um código que desenha a árvore.

2 Funcionamento

Ao ser iniciado, o programa deve carregar um arquivo de imagem. Para tanto, utilizaremos uma biblioteca simples (integrada no projeto) denominada *SOIL*.

1. Ler a imagem colorida, onde cada *pixel* (ponto da imagem) é representado em RGB (componente vermelho, verde e azul: cada um é um *unsigned char*).

2. Obter do usuário o **menor nível de detalhe** desejado. Esse valor varia de imagem para a imagem, e representa, em linhas, gerais, a menor diferença média entre os pixels de uma determinada região e a cor média da região (veja detalhes a seguir).

3. Gerar a *quadtree* e chamar a função para desenhá-la na tela (já fornecido).

O código fornecido (ver seção 3) contém um projeto com as bibliotecas necessárias para compilá-lo, mais algumas imagens de teste.

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, e pode-se usar as seguintes teclas:

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

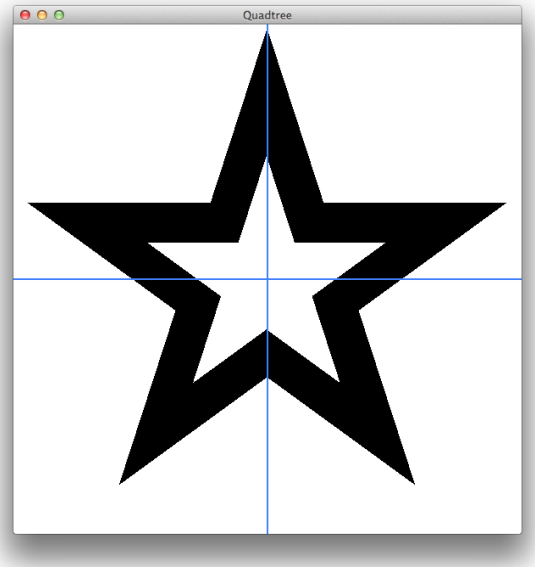
As próximas seções explicam como realizar as etapas.

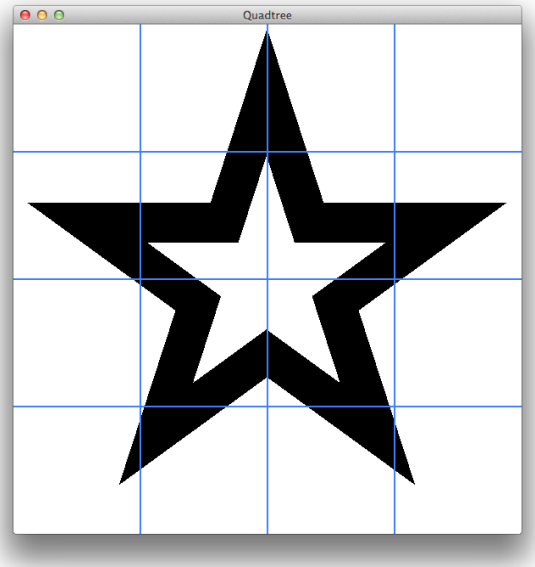
2.1 Geração da *Quadtree*

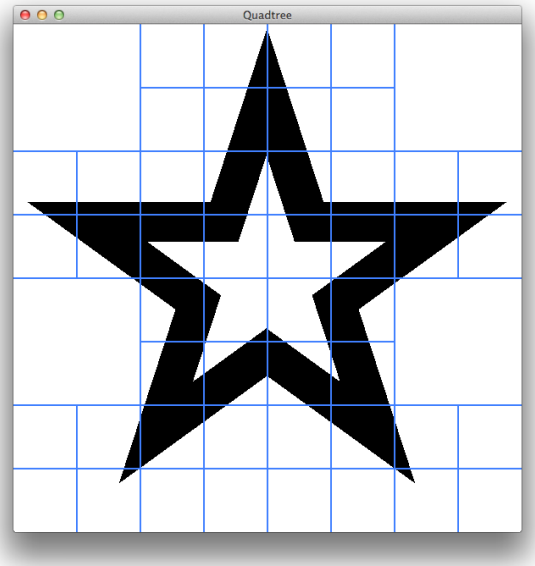
O processo de geração da *quadtree* é um **algoritmo recursivo**: a raiz da

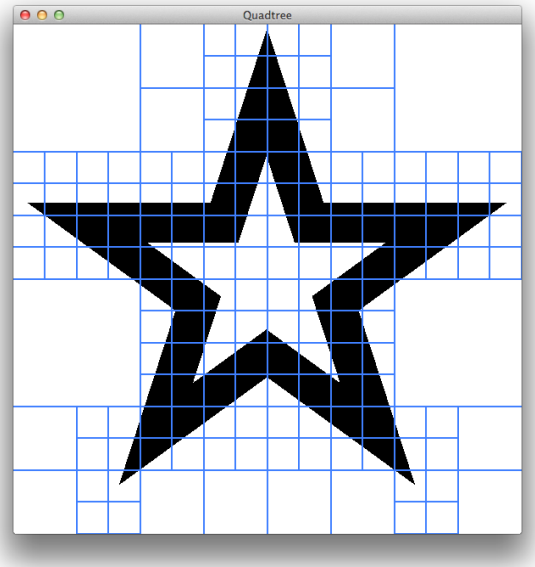
ão da imagem. Se essa região não tem muitos

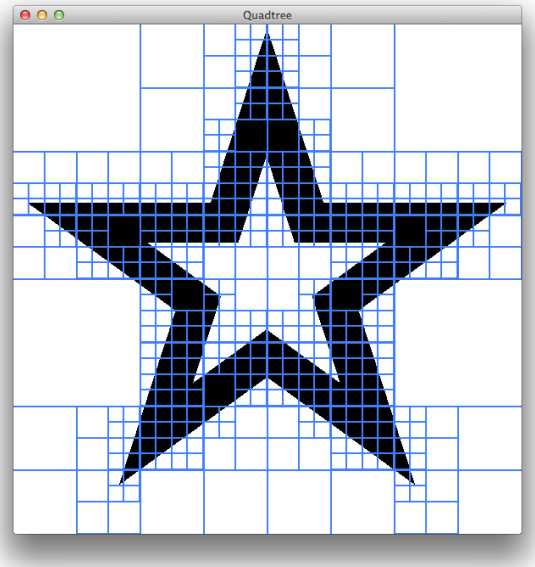
detalhes, o processo se encerra. Caso contrário, são gerados nodos filhos para cada subregião: superior direita (NW), superior esquerda (NE), inferior direita (SW) e inferior esquerda (SE). E o algoritmo é novamente aplicado para cada uma delas. Para entender o processo, veja a sequência de figuras abaixo, que mostra o resultado ao algoritmo para 1, 2, 3, 4 e 5 níveis na árvore:



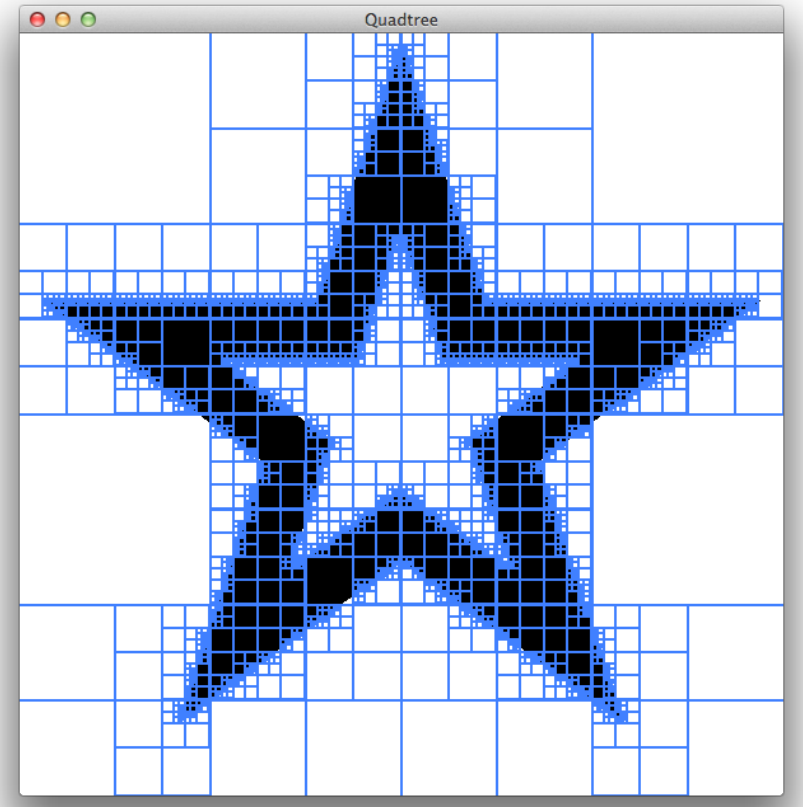




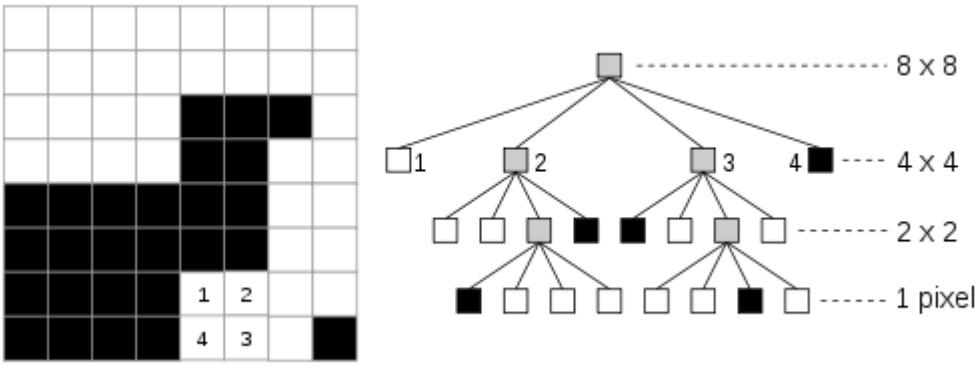




A última imagem mostra o resultado final, ou seja, ao chegar no nível de detalhe desejado. Esse nível é especificado através de um valor, que deve ser comparado com o nível de detalhe da região (ver seção 2.2): se este último for **inferior ou igual ao valor informado**, significa que o nível desejado foi atingido e o processo se encerra. Note que regiões com apenas uma cor têm nível de detalhe igual a **zero**.



Finalmente, a figura abaixo apresenta um exemplo de árvore gerada, para uma imagem simples:



2.2 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*:

```
typedef struct {
    int width, height;
    RGB *img;
} 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.

O módulo *quadtree.c*, na função *geraQuadtree*, demonstra como acessar as componentes de cor de cada pixel:

```
// pixels é um ponteiro que permite o acesso do vetor img como matriz -

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

Ou seja, estamos exibindo, em hexadecimal, as componentes de cor R, G e B dos primeiros 10 pixels (a partir do canto superior esquerdo).

2.3 Cálculo do nível de detalhe

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

1. Descobrir a **cor média** da região (média de todas as cores dos pixels da região, isto é, a média das componentes R, G e B de todos os pixels). A cor média é, portanto, também expressa em R, G e B.

2. Calcular a **diferença** entre cada pixel da região e essa cor média. Isso pode ser feito através da distância euclideana entre cada cor e a cor média , imaginando que as componentes R, G e B são coordenadas espaciais:

3. Acumular essa diferença ao longo de toda a região e calcular a **diferença média** no final. Para tanto, essa soma de diferenças precisa ser dividida pela área em pixels da região.

4. A diferença média resultante corresponde ao **nível de detalhe** da região.

2.4 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*, com a seguinte estrutura:

```
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;
```

A *struct Quad* **não deve ser alterada**, pois é usada dessa forma para desenhar a *quadtree*. Declaramos também 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);
```

3 Compilação

Download do código base: quadtree-base.zip

Este zip contém o projeto completo 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.

3.1 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*.

3.2 Outros ambientes ou terminal

Caso esteja usando outro ambiente de desenvolvimento, fornecemos 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
```

4 Avaliação

Leia com atenção os critérios de avaliação:

- Pontuação:
  - Cálculo da cor média de cada região: 2 pontos
  - Cálculo correto do nível de detalhe de cada região: 2 pontos
  - Geração correta da *quadtree*: 6 pontos
- Os trabalhos são **em duplas ou individuais**. Os arquivos contendo o código-fonte (.cpp) devem ser compactados em um arquivo .zip e submetidos 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 identado corretamente (qualquer editor decente **faz isso automaticamente**).
- **A cópia parcial ou completa do trabalho terá como conseqüência 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 idéia encontrada na rede for utilizada na implementação, sua descrição e referência deve constar no artigo.

Document generated by eLyXer 1.2.5 (2013-03-10) on 2020-10-24T15:29:57.545486