Trabalho Prático 1

Gerador de *ASCII Art*

09/2020

1 Introdução

O uso de caracteres da tabela ASCII para produzir desenhos não é uma idéia recente: vem de tempos longínquos, onde os dispositivos de saída eram monitores monocromáticos e impressoras matriciais ou de linha. Mais recentemente, se tornou uma forma de "arte" - daí o termo *ASCII Art*. Ou seja, imagens criadas exclusivamente com o uso de caracteres da tabela ASCII.

O objetivo deste trabalho é explorar os conceitos de programação C vistos em aula, criando um programa capaz de produzir a representação de uma imagem qualquer em ASCII. A saída do programa será na forma de um arquivo HTML, que poderá ser visualizado no browser. Se você estava em outro planeta nos últimos anos e nunca viu isso, há diversos serviços na rede que fazem esse tipo de conversão:

- http://www.text-image.com/
- http://picascii.com/
- http://lunatic.no/ol/img2aschtml.php

O resultado pode ser visto na figura 1:

Figura 1 Imagem original (esq.) e convertida para *ASCII Art* (dir.)

2 Funcionamento

Ao ser iniciado, o programa deve solicitar o nome de um arquivo de imagem e carregá-lo (ou obter o nome por parâmetros de linha de comando).

Para ler as imagens utilizaremos uma biblioteca simples (integrada no projeto de exemplo) denominada *SOIL*. Veja detalhes na seção 2.1. Após a leitura, a imagem deve ser processada e gerado o arquivo de saída em HTML.

Parte do objetivo deste trabalho é a criação de um algoritmo que produza um resultado interessante. Uma possível estratégia é a seguinte:

1. Obter do usuário o **nome da imagem** a ser carregada, e o **fator de redução desejado**. Por exemplo, o fator 50% utilizará apenas metade dos pixels originais na saída. Esse fator é usado para calcular o tamanho dos blocos de pixels (ver passo 5).

2. Ler a imagem colorida, onde cada pixel (ponto da imagem) é representado em RGB (ver seção 2.1).

3. Converter a imagem para tons de cinza, isto é, eliminar a cor e considerar que qualquer pixel pode ser representado por um único número inteiro, representando variações de preto (0) a branco (255).

4. Associar cada caractere a um tom de cinza específico. Você pode pesquisar nos sites indicados (ou outros) para ver como isso normalmente é realizado. Mas a idéia básica é que caracteres que ocupam mais espaço visualmente correspondam a cores mais claras (considerando um fundo preto). Por exemplo, "@" é mais "claro" do que "."

5. De acordo com as proporções das letras (que não são quadradas), agrupar os pixels da imagem em blocos retangulares (por exemplo, 4 x 5). Cada bloco irá se transformar em um único caractere na saída. Isso é necessário, pois as imagens normalmente são muito grandes para fazer uma correspondência 1:1.

6. Calcular o tom de cinza médio de cada bloco (por exemplo, fazendo a média de todos os pixels do bloco).

7. Para cada bloco de pixels, escolher e gerar um caractere na saída, cujo tom de cinza é uma boa aproximação para a média do bloco.

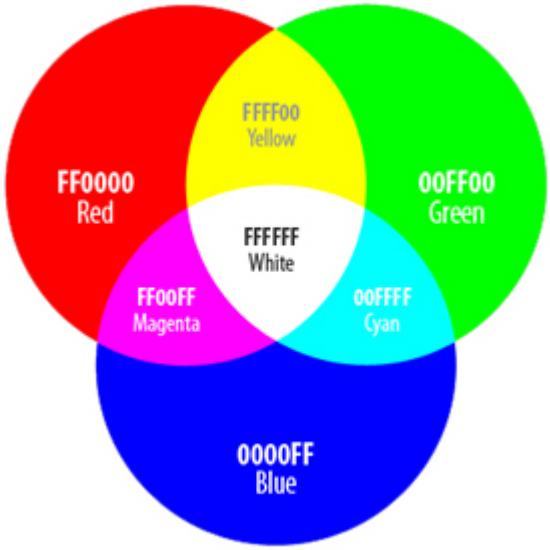
As próximas seções dão algumas dicas de como realizar certas etapas.

2.1 Leitura da imagem

Uma imagem é geralmente representada por uma matriz de pontos (*pixels*) onde cada cor é definida por 3 componentes: vermelho (R), verde (G) e azul (B). Cada uma dessas componentes usualmente é codificada em **um byte**, o

que produz **3 bytes por *pixel*** (24 bits) - ou seja, 16 milhões de possíveis cores. Em outras palavras, as intensidades (R, G, B) variam de 0 a 255, onde 0 é escuro e 255 é claro.

Veja abaixo como diversas cores são representadas nesse formato - cada cor está expressa pelas componentes RGB em hexadecimal.



FFFFFF	000000	333333	666666	999999	CCCCCC	CCCC99	9999CC	666
660000	663300	996633	003300	003333	003399	000066	330066	660
990000	993300	CC9900	006600	336666	0033FF	000099	660099	990
CC0000	CC3300	FFCC00	009900	006666	0066FF	0000CC	663399	CC(
FF0000	FF3300	FFFF00	00CC00	009999	0099FF	0000FF	9900CC	FF(
CC3333	FF6600	FFFF33	00FF00	00CCCC	00CCFF	3366FF	9933FF	FF(
FF6666	FF6633	FFFF66	66FF66	66CCCC	00FFFF	3399FF	9966FF	FF6
FF9999	FF9966	FFFF99	99FF99	66FFCC	99FFFF	66CCFF	9999FF	FF9
FFCCCC	FFCC99	FFFFCC	CCFFCC	99FFCC	CCFFFF	99CCFF	CCCCFF	FFC

O código fornecido define duas *structs*: uma para representar um *pixel* e outra para representar a imagem inteira. Após a leitura da imagem, os *pixels* estarão disponíveis no vetor *pic.img*

```
// Um pixel RGB (24 bits)
typedef struct {
    unsigned char r, g, b;
} RGB;
```

```
// Uma imagem RGB
typedef struct {
    int width, height;
    RGB* img;            // vetor de pixels
} Img;

// A imagem
Img pic;

// Cada pixel está armazenado em pic.img[i], onde i é o índice do pixel
// Exemplo: acessando os dois primeiros pixels
RGB aux1 = pic.img[0];
RGB aux2 = pic.img[1];
...
```

2.2 Conversão para tons de cinza

O processo de conversão de uma imagem para tons de cinza pode ser feito com o algoritmo descrito abaixo. A idéia é simplicar a imagem, deixando as três componentes R,G e B de cada ponto iguais entre si - equivale a "tirar a cor" da imagem. O valor a ser colocado nestes 3 componentes deverá ser igual à **intensidade da cor**, dada pela fórmula:

```
Para cada ponto (x,y) da imagem:
    Obtem a cor do pixel (r,g,b)
    // calcula o equivalente cinza da cor
    i = (0.3 * r + 0.59 * g + 0.11 * b)
    // armazena o cinza no pixel
    EscreveCor(x,y,i,i,i) # x,y,r,g,b
```

2.3 Geração do HTML de saída

Para gerar o código HTML resultante, é preciso escrever determinadas *tags* no arquivo de saída, de forma que o browser saiba como interpretar os caracteres:

```
<html><head></head>
<body style="background: black;" leftmargin=0 topmargin=0>
```

```
<style>
   pre  {
          color: white;
    font-family: Courier;
      font-size: 8px;
   }
</style>
```

Esse trecho define que o fundo será **preto**, enquanto os caracteres serão **brancos**, para o estilo pré-formatado (*tag PRE*) e com uma fonte bem pequena (*8px*). A partir deste ponto, a imagem deve aparecer dentro de um bloco *<pre>...</pre>*:

```
<pre>
... caracteres da imagem - linha 1 ...
... caracteres da imagem - linha 2 ...
...
</pre>
```

Finalmente, deve-se fechar o corpo da página e o HTML em si:

```
</body>
</html>
```

3 Código base e imagens de teste

O arquivo loader-t1-20202.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:
  - Conversão para cinza: 2 pontos
  - Obtenção do cinza médio do bloco: 2,5 pontos
  - Manutenção da proporção da "imagem" resultante: 2,5 pontos
  - Geração da saída em HTML: 2 pontos
  - Escolha do conjunto de caracteres para a saída: 1 ponto
- Os trabalhos são **em duplas ou individuais**. A pasta do projeto deve ser compactada em um arquivo .zip e este deve ser submetido pelo *Moodle* até a data e hora especificadas.
- Não envie .rar, .7z, .tar.gz - 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 verificação de cópias é feita inclusive entre turmas.**
- **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 2020-09-16T15:16:24.731807