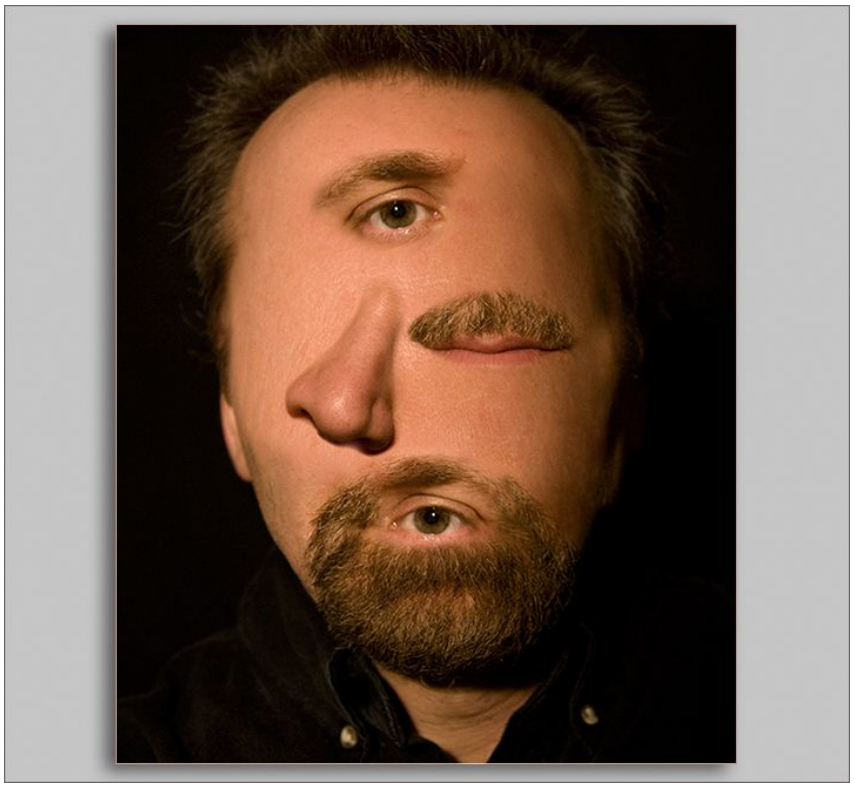
Introdução a Técnicas de Programação - 2019.1

Descrição de projeto

Processamento de Imagem



Introdução

Processamento de imagem é qualquer forma de processamento de dados no qual a entrada e saída são imagens tais como fotografias ou quadros de vídeo. Ao contrário do tratamento de imagens, que preocupa-se somente na manipulação de figuras para sua representação final, o processamento de imagens é um estágio para novos processamentos de dados tais como aprendizagem de máquina (inteligência artificial) ou reconhecimento de padrões (sua câmera detecta um rosto para a foto, por exemplo). A maioria das técnicas envolve o tratamento da imagem como um sinal bidimensional, no qual são aplicados padrões de processamento de sinal.

Para a realização do projeto, faz-se necessário apresentar alguns conceitos básicos de processamento de imagem, a começar como elas são representadas em arquivos e depois como essas representações podem ser manipuladas.

Representações de imagens

Imagens digitais são normalmente representadas de duas formas: vetorial ou *raster*. A primeira usa figuras geométricas, como polígonos, circunferências entre outros, para definir objetos vetoriais a serem usados na composição da imagem. A segunda forma é uma representação em que há uma correspondência de cada pixel da imagem a um conjunto de informações (como cor, transparência etc). Dizemos que esta última forma é uma representação "ponto-a-ponto" da imagem, normalmente realizada através de uma matriz onde cada célula contém a informação de um ponto, também conhecida como "mapa de bits" ou *bitmap*.

O tamanho da matriz define a resolução da imagem, normalmente especificada através da quantidade de colunas e de linhas. Por exemplo, a resolução 640 x 480 indica uma imagem cuja matriz possui 640 colunas e 480 linhas, ou seja que contém 307.200 pontos de informação.

Quando desejamos armazenar, transferir ou imprimir essa imagem, faz-se necessário armazená-la em um arquivo. A representação da imagem no arquivo não utiliza necessariamente a mesma representação da imagem em memória. Enquanto esta procura facilitar e tornar as operações de manipulação da imagem eficientes, aquela é normalmente voltada à compactação da imagem (tamanho do arquivo) com ou sem perdas de qualidade.

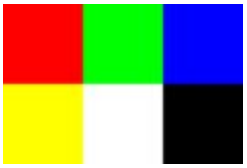
O projeto descrito neste documento se endereça a imagens do tipo *raster*. Sua representação na memória do computador é normalmente efetuado através de matrizes de pontos de informação, mas a representação em arquivo pode seguir diferentes formatos. Os formatos mais simples são os definidos pelo pacote NetPBM, encontrados nos ambientes Linux. O pacote NetPBM define alguns formatos, como o PBM (Portable Bitmap), PGM (Portable Graymap) e PPM (Portable Pixmap). Este último será utilizado no presente projeto.

Formato de imagem PPM

O formato de imagem PPM possui duas versões: uma textual e outra binária. A versão textual armazena os valores de cada pixel em um arquivo ASCII. Cada pixel é definido por três componentes, indicando a intensidade das cores vermelho, verde e azul. Ou seja, a cor do pixel é resultante da combinação das cores vermelho, verde e azul. Para entender como funciona essa combinação, você pode acessar http://www.rapidtables.com/web/color/RGB_Color.htm e variar os campos R (Red), G (Green) e B (Blue). A cor preta, por exemplo, é representada por R = 0, G = 0 e B = 0. Já o amarelo é representado por R = 255, G = 255 e B = 0.

O arquivo PPM obedece a uma formação específica composta por um cabeçalho, com informações sobre a imagem em geral, seguido de uma sequência de triplas RGB para cada um dos pixels. A função do cabeçalho é identificar que se trata de uma imagem PPM e de sua versão, as dimensões da imagem e o valor máximo de cada cor.

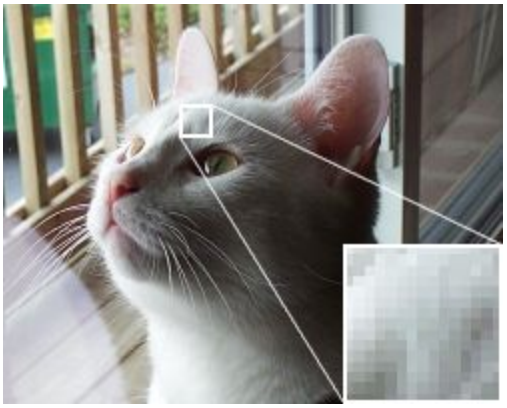
Para ilustrar melhor, a figura e conteúdo a seguir exemplificam uma imagem (ampliada para ser melhor exibida) com dimensões de 3x2 representada no formato PPM, versão textual.



```
P3
3 2
255
255 0 0
0 255 0
0 0 255
255 255 0
255 255 255
0 0 0
```

A primeira linha contém o identificador `P3`. Esse identificador serve para os programas que leem arquivos de imagem saber que o conteúdo a seguir adotam o formato textual do PPM. Em seguida, os valores 3 e 2, separados por espaço, indicam que a imagem tem 3 colunas e 2 linhas. Na terceira linha está o valor 255. Esse valor é usado para informar o valor máximo de cada componente do pixel e indica, de certa forma, o nível de qualidade da imagem. Valores baixos representam imagens com pouca qualidade e valores maiores representam maior qualidade. Por exemplo, se esse valor máximo for 4, ou seja cada componente pode assumir 5 valores (intervalo de 0 a 4), significa que a imagem pode ter no máximo 75 cores diferentes (5x5x5). Se o valor for 255, ou seja cada componente pode assumir 256 valores, teremos 16.777.216 possíveis cores para representar cada pixel.

Para ilustrar essa diferença, a primeira figura abaixo usa o valor padrão de qualidade 255, enquanto que a segunda figura usa como valor 16. Compare a diferença nas qualidades entre as figuras abaixo para notar a diferença.



As demais linhas do arquivo se referem aos componentes RGB dos pixels da imagem. A ordem em que os pixels se encontram seguem linha a linha. Ou seja, em uma imagem de 3 colunas e 2 linhas, os três primeiros pixels correspondem à primeira linha e os três seguintes à segunda linha.

Neste trabalho, considere os arquivos de imagem no formato PPM versão textual (P3) com o valor de qualidade fixo em 255 (cada componente pode variar de 0 a 255). As dimensões podem, entretanto, variar.

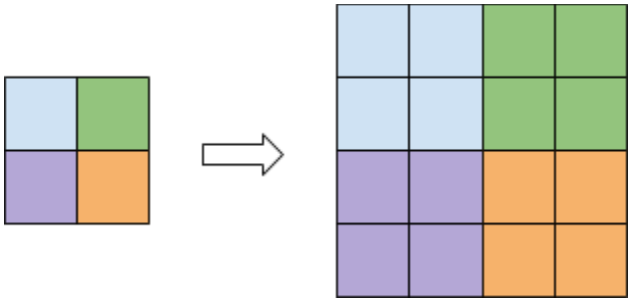
Transformação de imagens

Existem inúmeros métodos de transformação de imagens. As solicitadas neste trabalho são: 1) transformação de uma imagem colorida em.escala de cinza, 2) ampliar, reduzir e rotacionar imagens, 3) aplicar filtros de *blurring*, *sharpening* e detecção de bordas. Outros filtros e transformações podem ser implementados no trabalho como elementos extras.

Há várias abordagens para se transformar uma imagem colorida em escala de cinza. Seja qual for a abordagem escolhida, o resultado irá transformar em cada pixel componentes RGB com diferentes valores (ex: R = 235, G = 102 e B = 188) em componentes RGB com o mesmo valor (ex: R = 150, G = 150 e B = 150). O fato dos componentes terem valores iguais indica a não predominância de uma cor sobre as outras, ou seja cinza.

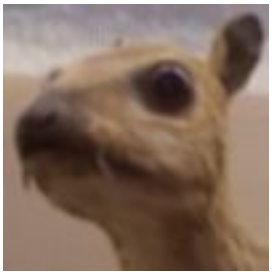
Outra forma de transformar a imagem é complementar à transformação em escala de cinza. Consiste em segmentar as imagens em regiões binárias (preto ou branco) a partir de um valor limite na escala de cinza (do método anterior), também conhecido de *thresholding*. Nesse método, os pixels são selecionados e tratados como objetos, atribuindo-se valores para cada de acordo com os níveis de cinza, valores abaixo do *threshold* (valor limite) são considerados 'elemento de fundo' (preto) e valores igual ou acima deste limite são considerados como parte do objeto de interesse (branco). Existem várias maneiras de inferir o *threshold* que irá segmenta a imagem em uma imagem binária. Pode ser a simples aplicação de um valor fixo (pré-definido) ou através de métodos iterativos, nos quais são aplicados valores aleatórios do limite, percorrendo várias vezes a imagem, segmentando-a e armazenando tais valores, que por fim será tirada uma média para um novo valor limite.

Ampliar, reduzir e rotacionar uma imagem envolve a definição de uma nova imagem com mais ou menos pixels. Na ampliação em 2x de uma imagem, por exemplo, cada pixel da imagem antiga terá 4 pixels na nova, como exemplificado abaixo. A redução segue o caminho inverso, fazendo com que vários pixels (inclusive de cores distintas) sejam representados por um único pixel. A rotação segue princípio similar, porém alterando a posição das cores dos pixels em função da rotação (90º à esquerda, 90º à direita ou 180º). Vale salientar que numa rotação de 90º o número de colunas e linhas se alternam. Por exemplo, ao rotacionar 90º uma imagem 640x480, o resultado será uma imagem 480x640.



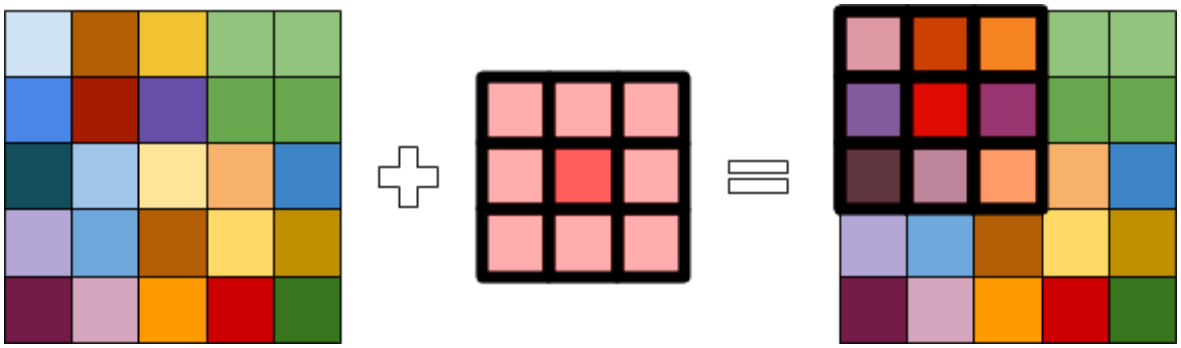
Aplicação de filtros é uma técnica de processamento de imagens que normalmente manipula os valores de uma vizinhança para modificar a imagem. Esta técnica é normalmente utilizada para destacar, suavizar e/ou remover determinadas características da imagem. Podemos, por exemplo, aumentar o contraste dos pixels vizinhos ou deixá-los mais difusos, como ilustrados nas imagens a seguir, em que os filtros de *sharpening* e *blurring* foram aplicados.

Sharpening



Blurring

Para a aplicação de filtros, é necessário utilizar uma "máscara" que percorre toda a imagem realizando as operações sobre os pixels. Essa máscara, também chamada de *kernel* ou matriz de convolução, é definida por uma matriz cujos valores vão atuar sobre os pixels da imagem. Normalmente, as matrizes são quadradas de tamanho ímpar (ex: 3x3, 5x5, 7x7…) indicando a área de influência dos vizinhos sobre um pixel. Ao aplicar um filtro 3x3, estamos indicando que os pixels adjacentes a um determinado pixel irão influenciar sua nova cor. A figura abaixo ilustra o primeiro passo da aplicação de uma máscara 3x3 em uma imagem 5x5. A imagem resultante desse primeiro passo mostrar que os pixels vizinhos ao pixel de coordenada (1,1) afetaram o cor final do mesmo. Após esse primeiro pixel, o processo continua realizando o mesmo procedimento para todos os demais pixels.

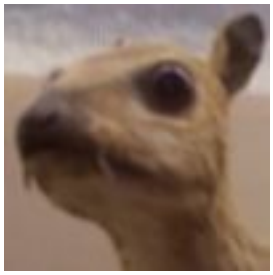


Em função dos valores na matriz e das operações realizadas, podemos transformar a imagem de diferentes maneiras. As matrizes a seguir e suas correspondentes imagens ilustram os resultados na aplicação dos filtros de *sharpening*, *blurring* e detecção de bordas.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Sharpening

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Blurring

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Detecção de bordas

Descrição do projeto

O projeto de Processamento de Imagem deve ser desenvolvido em linguagem C, a ser executado, em sua versão mais simples, através de linha de comando (entrada e saída em um console/terminal). Além dos filtros básicos para o processamento de imagens, o projeto desenvolvido deve implementar os seguintes requisitos funcionais:

- Será desenvolvido um programa que lê um arquivo PPM e realiza uma ou mais operações de transformação da imagem digitadas pelo usuário.
- Os comandos possíveis são:
    - ‘thr’: binarização da imagem usando thresholding
    - ‘blu’: executa blurring
    - ‘sha’: executa sharpening
    - 'rot': rotação da imagem, dado o ângulo
    - 'amp’: ampliar a imagem, dado o zoom.

    - ‘red’: reduzir a imagem, dado o zoom.
- O término do programa ocorre em uma das seguintes situações:
    - a segmentação foi feita completamente
    - aconteceu um erro na segmentação da imagem
    - o usuário interrompeu o programa.

O projeto deve atender também os seguintes critérios de programação:

1. **Uso de arranjos / matrizes;**
2. **Uso de registros (`struct`);**
3. **Definição de novos tipos de dados através de `typedef`;**
4. **Uso de alocação dinâmica;**
5. **Pelo menos uma função/procedimento que usa recursão;**
6. **Leitura de arquivos;**
7. **Modularização do programa em diferentes arquivos (uso de diferentes arquivos .c e .h, cada um com sua funcionalidade);**
8. **Definição de um padrão de indentação do código fonte e de nomenclatura das sub-rotinas e variáveis, e a adequação a este padrão;**
9. **Documentação adequada do código-fonte.**

Observações:

- O projeto deve ser desenvolvido individualmente ou em duplas (grupos de dois alunos). Não serão permitidos grupos com três ou mais alunos.
- Para facilitar o acompanhamento do projeto, cada dupla deve estar associada a uma única turma de ITP. Ou seja, não serão permitidas duplas formadas por alunos matriculados em turmas de ITP diferentes.
- Cada dupla deve desenvolver sua solução de forma independente das demais. Soluções idênticas ou parecidas serão consideradas plágios e, portanto, sanções serão devidamente aplicadas em todas as duplas com soluções similares.
    - https://sigaa.ufrn.br/sigaa/public/programa/noticias_desc.jsf?lc=pt_BR&id=412&noticia=105986912
- Códigos e algoritmos podem ser utilizados da web desde que devidamente referenciados. Caso sejam encontrados trechos de código na web equivalentes aos apresentados pelo grupo sem a devida citação, o código será igualmente considerado plágio e sanções serão aplicadas ao grupo. Vale salientar que a avaliação será realizada unicamente sobre o código produzido pela dupla. Códigos retirados da web, apesar de permitidos com a devida citação, serão desconsiderados dos critérios de pontuação.

Avaliação

Esta avaliação compreende a execução e desenvolvimento do projeto em apresentações. A cada dia, o grupo deve apresentar uma etapa do projeto desenvolvido, seguindo o calendário abaixo:

ETAPA 1 - 10/06/2019 (modificado)

1. Tipos de dados necessários (`typedef`, `struct` e `enum`);

2. Modularização do programa (quais os arquivos .c e .h)
3. Leitura de imagens de arquivo;
4. Transformar em escala de cinza

ETAPA 2 - 12/06/2019 (modificado)

5. Transformações com as imagens: ampliar, reduzir, rotacionar.

ETAPA 3 - 17/06/2019

6. Subrotinas de *blurring* e *sharpening*;
7. Detecção de bordas;

ETAPA 4 - 19/06/2019

8. Implementação de elementos extras, definidos pelo próprio grupo.

ETAPA Extra - 21/06/2019

Entrega de de qualquer parte que não tenha sido apresentada. Último dia!

Sobre as duplas

Os alunos têm ATÉ o dia **10 de maio de 2019** para comunicar aos professores se farão o trabalho em dupla (e a composição da mesma) ou se farão individualmente.

Critérios de pontuação

O desenvolvimento do projeto aqui descrito vale **100%** da nota da terceira unidade.

A pontuação da avaliação seguirá os critérios e distribuição abaixo:

- **Atendimento dos requisitos funcionais: 50%**
  a imagem está representada corretamente? os filtros estão implementados corretamente? os filtros podem ser aplicados a qualquer imagem? etc.
- **Uso dos recursos da linguagem C: 20%**
  a dupla demonstrou saber usar de forma adequada os recursos da linguagem C (arranjos, structs, typedefs, recursividade, etc)?
- **Organização do código e documentação: 10%**
  o código está documentado? a identação e uso de { } seguem um padrão **(identação)**? o programa está devidamente modularizado em diferentes arquivos?
- **Funcionalidades extras: 20%**
  as funcionalidades extras desenvolvidas pela dupla foram suficientemente complexas?

A pontuação a ser dada pelas funcionalidades extras não é definida *a priori*. Cada caso será avaliado em função da complexidade envolvida. Itens extras de baixa complexidade serão desconsiderados na pontuação.

Entrega do projeto

O projeto deve ser submetido pelo SIGAA até a data **21 de junho de 2019** em um **arquivo comprimido (.zip)** contendo os arquivos fontes do projeto (.c e .h) e um arquivo README.TXT. Este arquivo deve ter as seguintes informações:

- O que foi feito (o básico e as funcionalidades extras implementadas);
- O que não foi feito (caso não tenha sido possível implementar alguma funcionalidade);

- O que seria feito diferentemente (quando aprendemos à medida que vamos implementando, por vezes vemos que podíamos ter implementado algo de forma diferente. Assim, esse tópico é para vocês refletirem sobre o que vocês aprenderam durante o processo que, se fossem fazer o mesmo projeto novamente, fariam diferente);
- Como compilar o projeto, deixando explícito se foi utilizada alguma biblioteca externa que precise ser instalada, que versão e quais parâmetros extras são necessários para o compilador.
- Em caso de duplas:
    - Identificação dos autores;
    - Contribuição de cada integrante no desenvolvimento do projeto (quem fez o quê).

Recomendações diversas:

1. Solução de back-up: não será tolerada a desculpa de que um disco rígido falhou nas avaliações. Assim, é importante manter várias cópias do código-fonte desenvolvido ou usar um sistema de backup automático como o Dropbox, Google Drive, Box ou similares. Uma solução melhor ainda é fazer uso de um sistema de controle de versões como git, e um repositório externo como Github ou Bitbucket.
2. Especificar precisamente a interface e o comportamento de cada sub-rotina. Usar esta informação para guiar o desenvolvimento e documentar o código desenvolvido.

Extras

- Implementação de filtros mais avançados: Fourier, Gauss, detecção de objetos específicos na imagem, etc.
- Trabalhar com desenhos de linhas e curvas Bezier.
- Desenho de fontes. A ideia é usar curvas de Bezier para desenhar fontes como TTF, por exemplo. Um exemplo de chamada seria plot_string(x, y, "string");
- Compressão de imagens: usar um algoritmo como o RLE para criar imagens compactadas simples. (https://en.wikipedia.org/wiki/Run-length_encoding)
- Desenho de gráficos de funções. Ler uma função e plotar a mesma. Seria usado structs para representar as funções. Uso de recursão para implementar um parser simples.