INF101 – Introdução à Programação II

Roteiro de Prática: 4 de fevereiro de 2020

Introdução

O objetivo principal desta prática é exercitar o uso de funções para estruturar a implementação de um programa em Python e também a utilização da estrutura de dados *arranjo bidimensional*. Para tanto, vamos repetir o assunto sobre processamento de imagem que foi usado em INF100, no semestre passado. Porém vamo-nos ater apenas à solução da operação de *image embossing*. Segundo a Wikipedia, *image embossing* é uma técnica de computação gráfica em que cada pixel de uma imagem é reposto por um realce ou uma sombra, dependendo das fronteiras claro-escuro da imagem original. Áreas de baixo contraste são repostas por um fundo cinza. Assim, a imagem filtrada representará a taxa de mudança de cor de cada ponto da imagem original. Ao aplicar um filtro de *embossing* a uma imagem, normalmente, resultará em uma imagem parecendo com gravação em relevo em papel ou metal da imagem original, daí o nome (em inglês).

Antes de obter o efeito proposto, temos que primeiro transformar a imagem, se colorida, em tons de cinza. Para que o código do programa fique bem estruturado, vamos criar e usar funções apropriadas tanto para o efeito de tons de cinza quanto para o efeito de *embossing*.

Algoritmo para efeito de tons de cinza

Reponha cada componente `(r, g, b)` da imagem pela luminância (brilho) do respectivo pixel:

```
luminancia = int(0.299 * r + 0.587 * g + 0.114 * b)
```

Lembre-se de que cada pixel `(r, g, b)` é obtido do elemento `im[y][x]` da matriz `im` contendo a imagem colorida original. Use os atributos `im.altura` e `im.largura` para obter as dimensões da imagem. Esses atributos são disponibilizados pelo arquivo fonte `imagens.py` que você deverá obter do *site* de entrega de trabalhos.

Algoritmo para efeito de *embossing*

Agora, ao invés de alterar cada pixel de forma independente, usaremos uma pequena matriz, chamada de *matriz* ou *filtro de convolução*, que será "passada" em toda a imagem, transformando cada pixel de acordo não só com o valor dele, mas também com o de seus vizinhos. A matriz que será usada é a seguinte:

```
filtro = [[-1, -1,  0],
          [-1,  0,  1],
          [ 0,  1,  1]]
```

Cada bloco de 3 x 3 pixels da imagem será multiplicado por esse filtro para alterar apenas o pixel "central" da matriz de cada vez, como ilustrado abaixo:

3	101	36	5	2
251	117	139	184	40
134	13	57	82	243
186	250	10	75	253

×

-1	-1	0
-1	0	1
0	1	1

		-102		

113	59	55	64	143

Antes de começar o processamento dos pixels, precisamos fazer uma cópia da imagem para armazenar os pixels transformados. Isso pode ser feito com o comando:

```
im2 = im.copia()
```

que cria uma cópia da imagem armazenada em **`im`** e atribui à variável **`im2`**. Faremos isso só para ter outra imagem com as mesmas dimensões e características da original. Iremos então pegar cada pixel de **`im`**, processá-lo em relação a seus vizinhos usando o filtro acima, e colocar o resultado em **`im2`**. Isso pode ser feito da seguinte forma:

```
# Percorre todas as coordenadas da matriz original, deixando de fora as bordas.
for y in range(1, im.altura-1):
    for x in range(1, im.largura-1):
        pix = 0  # inicia somatório
        # Percorre matriz de convolução.
        for i in range(0, 3):
            for j in range(0, 3):
                # Pega pixel "embaixo" da coordenada do filtro.
                r, g, b = im[y-(1-i)][x-(1-j)]
                # Multiplica pelo filtro e soma. Para isso podemos usar apenas
                # um dos componentes (r, g ou b), já que todos são iguais.
                pix = pix + r * filtro[i][j]
        # Corrige o brilho e garante que fique entre 0 e 255.
        pix = max(0, min(255, pix+128))
        # Atribui o pixel a im2.
        im2[y][x] = (pix, pix, pix)
```

Observe a linha:

```
pix = max(0, min(255, pix+128))
```

Ela nada mais é do que a sequência:

```
pix = pix + 128
if pix > 255:
    pix = 255
elif pix < 0:
    pix = 0
```

que pode ser reduzida para:

```
pix = min(255, pix+128)
pix = max(0, pix)
```

e, finalmente, para:

```
pix = max(0, min(255, pix+128))
```

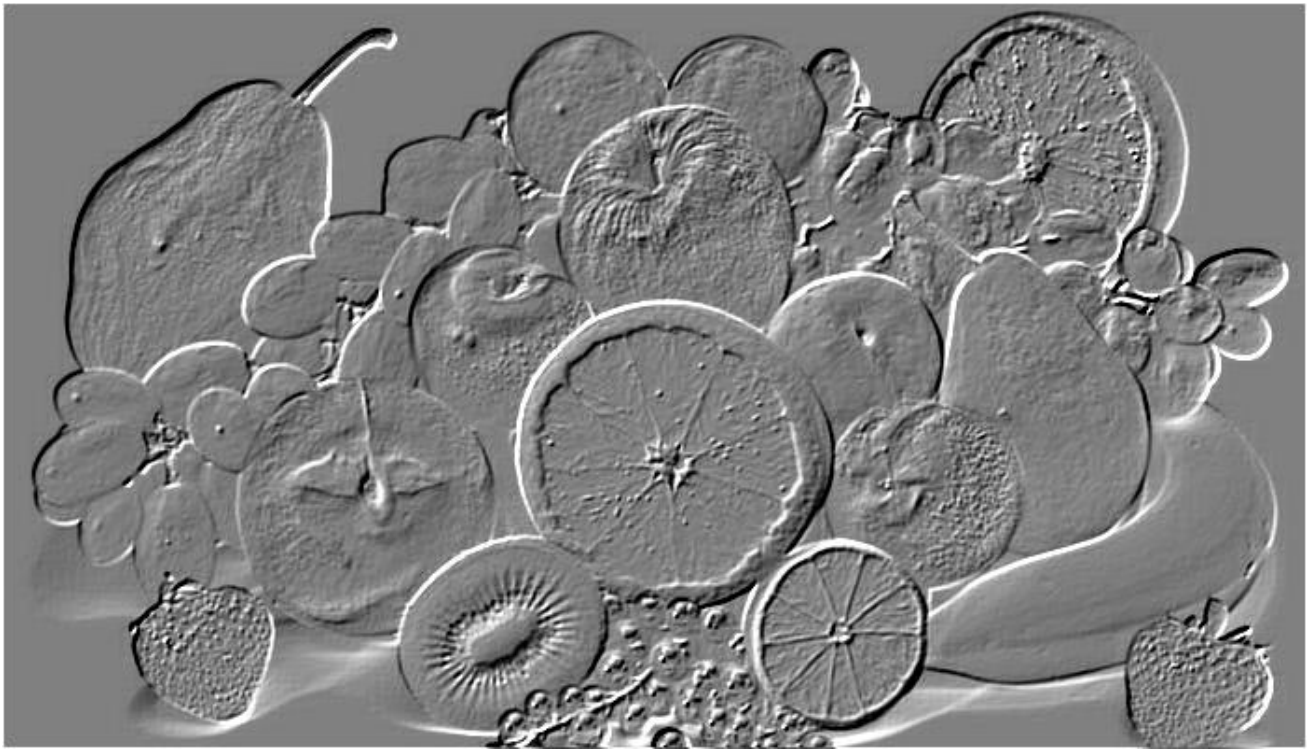
Depois de fazer todo o processo acima, retorne a imagem `im2`.

É necessário ter um pouco de paciência, pois essa operação demora um bom tempo, uma vez que, para cada pixel, precisamos fazer um somatório com todos os seus vizinhos. Apenas fique atento ao

`Python Shell` para ver se aparece alguma mensagem de erro enquanto você aguarda o processamento.

Ao aplicar o filtro de convolução dado à imagem abaixo:

obteremos a respectiva imagem *embossed* a seguir:



Instruções

1. Faça o *download* dos arquivos fontes `imagens.py` e `p01_esqueleto.py` e do arquivo de imagem `fruits-700.jpg` para seu diretório de trabalho, certamente, o diretório `/home/alunos`, mas pode ser qualquer um dentro do seu diretório *home*.
2. Modifique o nome do arquivo fonte `p01_esqueleto.py` para apenas `p01.py`. Não mexa no arquivo `imagens.py`. Deixe-o intacto no mesmo diretório de trabalho.
3. Abra a IDE IDLE, clicando duas vezes no respectivo ícone na área de trabalho.
4. Abra o arquivo fonte `p01.py`. Este contém ainda só o esqueleto do trabalho a ser entregue. As funções `tonal()` e `emboss()` precisam ser implementadas por você. Para tanto, use os respectivos algoritmos que foram apresentados no início deste roteiro.
5. Complete os códigos das funções pedidas nos locais apropriados onde há reticências (...). Remova as reticências antes de tentar executar sua implementação. Não modifique a função `main()` nem a chamada dela no final do código fonte.
6. Retire todos os erros de sua implementação e teste-a. Se seu programa estiver funcionando bem, vão aparecer a imagem original `fruits-700.jpg` e, depois de muitos segundos, a imagem *embossed*.

☞ Não se esqueça de preencher o cabeçalho do código fonte com seus dados e uma breve descrição do programa.

Após certificar-se de que seu programa esteja correto, envie o arquivo do programa fonte (`p01.py`) através do sistema de entrega do LBI.