UNIVERSIDADE FEDERAL DE SÃO CARLOS - UFSCAR
CIÊNCIA DA COMPUTAÇÃO
PROCESSAMENTO DIGITAL DE IMAGENS

RELATÓRIO: PREENCHIMENTO DE BORDA

São Carlos, Setembro de 2020

Discentes:

Agustin Gabriel Amaral Castillo, 592862

Jorge Vinicius Gonçalves, 758594

Thiago de Moraes Teixeira, 760667

Victoria de Martini de Souza, 75937

Docente:
Cesar Henrique Comin, Prof.
Universidade Federal de São Carlos – UFSCar

INTRODUÇÃO

Neste trabalho será apresentado a implementação de duas técnicas de preenchimento de borda, sendo elas:

- Mais próximo;
- Espelhado.

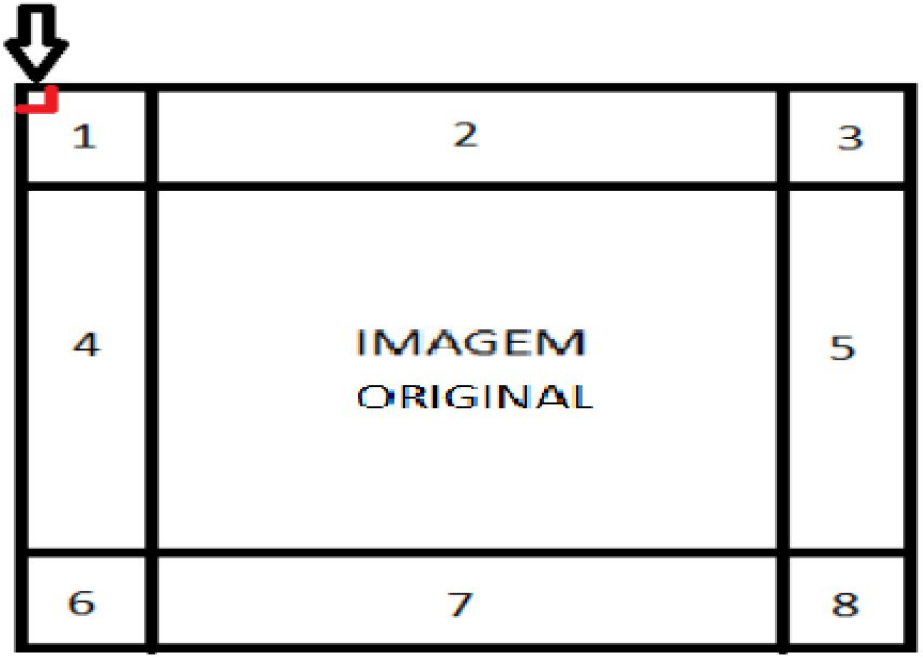
A técnica “mais próximo” está apresentada no arquivo “proximo.py”, o funcionamento dela consiste em pegar o último pixel da borda e extrapolar que todos valores para fora são o mesmo do pixel, deixando eles constantes.
A técnica “espelhado” está apresentada no arquivo “espelhado.py”, ela consiste em espelhar os valores da borda, supondo que pixel da borda esteja na coluna N, o pixel N+1 receberá o valor de N-1, o N+2 receberá o valor de N-2 e assim por diante.
Dentre essas e as outras técnicas existentes não é possível definir qual a melhor delas, pois cada uma se enquadra melhor para cada tipo de imagem, depende do programador observar a imagem que está sendo analisada e perceber qual delas se enquadra melhor.

CÓDIGO

1.1 Arquivo proximo.py

Tendo **img[numLinhas, numColunas],** sendo **numLinhas, numColunas** respectivamente os valores de linhas e colunas da imagem original, **tamanho_borda** o número de pixels desejados para o tamanho da borda e **img_border[numLinhas+tamanho_borda*2, numColunas+tamanho_borda*2]** sendo os pixels da nova imagem com a borda desejada**:**

- Na primeira técnica implementada (preenchimento com o pixel mais próximo) varrendo a imagem a partir do primeiro pixel, para preenchimento do valor de cada pixel, foi-se imaginado os seguintes cenários(1,...,8) para preencher a borda, com a varredura começando no primeiro pixel da nova imagem **img_border**[0,0] (Sinalizado na imagem guia abaixo):



- Cenários foram colocados em ordem que são verificados no código:

1. A posição do pixel selecionado for menor(ou igual) do que o tamanho da borda em linhas e colunas, portanto seu valor é:

```
if row<=tamanho_borda:
    if col<=tamanho_borda:
        img_border[row, col] = img[0, 0]
```

3. A posição do pixel selecionado for menor(ou igual) que o tamanho da borda em número de linhas e maior(ou igual) que o tamanho da borda em número de colunas, tendo seu valor atribuído como:

```
if row<=tamanho_borda:
    elif col>=numColunas+(tamanho_borda-1):
        img_border[row, col] = img[0, numColunas-1]
```

2. A posição do pixel selecionado for menor(ou igual) do que o tamanho da borda em linhas e estar nas colunas dentro do range da imagem, tendo seu valor atribuído como:

```
if row<=tamanho_borda:
    else:
        img_border[row, col] = img[0, col-tamanho_borda]
```

6. A posição do pixel selecionado for maior(ou igual) que o *numLinhas*+(tamanho_borda-1) em número de linhas e menor(ou igual) que o tamanho da borda em número de colunas, tendo seu valor atribuído como:

```
elif row>=numLinhas+(tamanho_borda-1):
    if col<=tamanho_borda:
        img_border[row, col] = img[numLinhas-1, 0]
```

8. A posição do pixel selecionado for maior(ou igual) que o *numLinhas*+(tamanho_borda-1) em número de linhas e

>=*numColunas*+(tamanho_borda-1) em colunas, tendo seu valor atribuído como:

```
elif row>=numLinhas+(tamanho_borda-1):
    elif col>=numColunas+(tamanho_borda-1):
        img_border[row, col] = img[numLinhas-1, numColunas-1]
```

7. A posição do pixel selecionado for maior(ou igual) que o *numLinhas*+(tamanho_borda-1) e as colunas dentro do range da imagem, tendo seu valor atribuído como:

```
elif row>=numLinhas+(tamanho_borda-1):
    else:
        img_border[row, col] = img[numLinhas-1, col-tamanho_borda]
```

4. A posição do pixel selecionado for menor(ou igual) que o número de colunas da borda e as linhas dentro do range da imagem, tendo seu valor atribuído como:

```
else:
    if col<tamanho_borda:
        img_border[row, col] = img[row-tamanho_borda, 0]
```

5. A posição do pixel selecionado for maior(ou igual) que o número de colunas da borda e as linhas dentro do range da imagem, tendo seu valor atribuído como:

```
else:
    elif col>num_cols+(tamanho_borda-1):
        img_border[row, col] = img[row-tamanho_borda, numColunas-1]
```

1.2 Arquivo espelhado.py

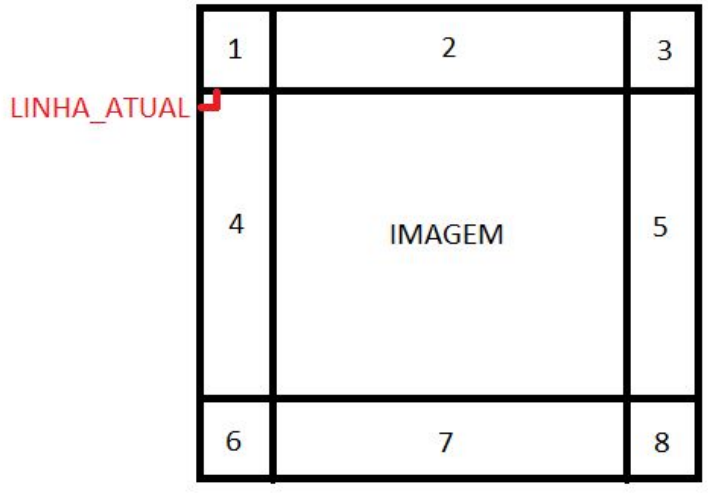
Tendo **img[numLinhas, numColunas],** sendo **numLinhas, numColunas r**espectivamente os valores de linhas e colunas da imagem original, **tamanho_borda** o número de pixels desejados para o tamanho da borda e **img_border[numLinhas+tamanho_borda*2, numColunas+tamanho_borda*2]** sendo os pixels da nova imagem com a borda desejada**:**

- Na segunda técnica implementada (preenchimento espelhado de valores dos pixels). Primeiramente foram pensados os mesmos 8 cenários da primeira técnica, porém agora adicionando +2 cenários em cada diagonal (cenários 1, 3, 6, 8) que serão explicitados logo abaixo.
- Variáveis relevantes do código:

linha_atual in range(tamanho_borda, num_rows+tamanho_borda): Irá começar varrendo na primeira linha que contém os pixels da imagem original, pois a borda, em pixels, vai de (0, *tamanho_borda*-1) e termina na última linha da imagem original, e será usado como base para direcionar o pixel selecionado para o espelhamento começar. Apontado na imagem abaixo.

diagonal_pixels = tamanho_borda: Será usado nas 4 diagonais, para orientar o espelhamento das linhas e colunas dentro das diagonais, que diminuem em 1 a cada pixel da diagonal que é lido, com isso é subtraído 1 de *diagonal_pixels* a cada iteração dentro do *for* da variável *pixel* e resetado na próxima execução com o novo valor de *pixel*. A condição no código para ser usada é ser maior ou igual a 2 (*diagonal_pixels*>=2) pois ela não é utilizada no último pixel da diagonal pois ele é único, não existindo mais pixels além dele para nenhum lado.

pixel in range(1, tamanho_borda+1): Será usado para percorrer o número exato de pixels desejados para o tamnho da borda, assim percorrendo tanto para dentro da imagem original, quanto para fora(borda) simultaneamente. Começa em 1, pois será usado em subtrações onde o 0(zero) não teria efeito.



-

- Cenários:

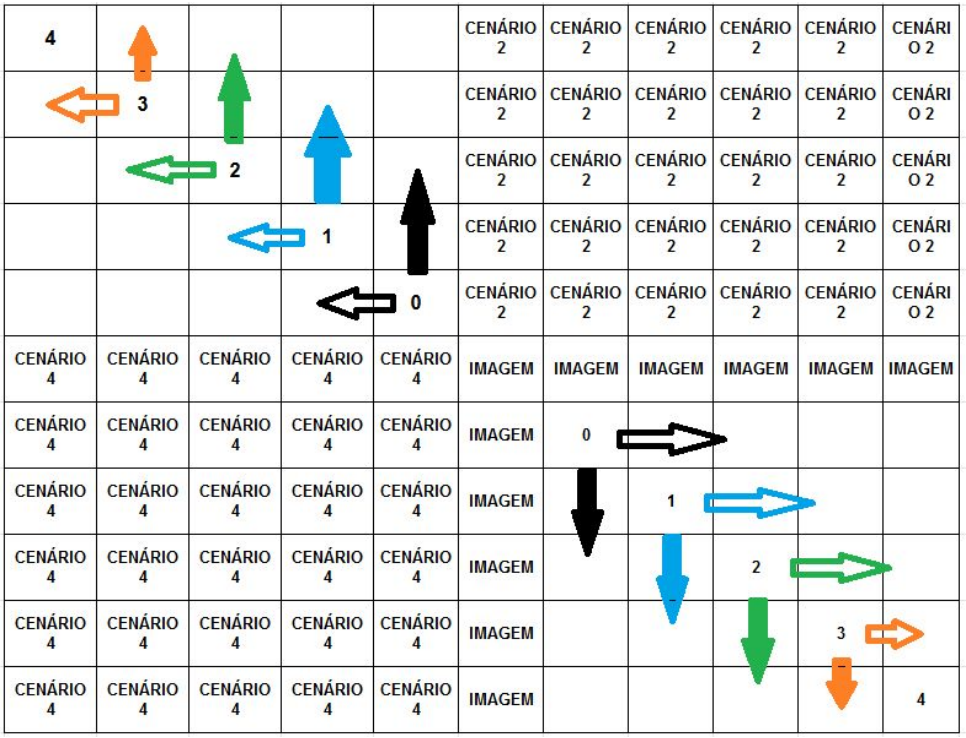
  2, 7. Usam a mesma lógica da primeira técnica, porem com o espelhado os novos valores nos cenários 2, 7 respectivamente ficarão:

```
# Direção vertical cima e baixo da imagem
img_border[tamanho_borda-pixel][linha_atual]                    = img[pixel][linha_atual-tamanho_borda]
img_border[numLinhas+(tamanho_borda-1)+pixel][linha_atual]      = img[numLinhas-1-pixel][linha_atual-tamanho_borda]
```

  4, 5. Usam a mesma lógica da primeira técnia, porém com o espelhamento os novos valores nos cenários 4, 5 respectivamente ficarão:

```
# Direção Horizontal esquerda e direita da imagem
img_border[linha_atual][tamanho_borda-pixel]                    = img[linha_atual-tamanho_borda][pixel]
img_border[linha_atual][numColunas+(tamanho_borda-1)+pixel]     = img[linha_atual-tamanho_borda][num_cols-1-pixel]
```

  1, 3, 6, 8. Dentro de cada diagonal temos a mesma lógica da primeira técnica porém agora adicionando as linhas e colunas a cada pixel da diagonal, que precisam ser espelhados, com isso teremos a seguinte configuração**:**
  Como exemplo pegamos a diagonal superior esquerda(cenário 1) e temos uma imagem com *tamanho_borda*=5 pixels, onde temos que espelhar a diagonal em sí, e também a linha/coluna relacionada a cada pixel da diagonal. Tais linhas e colunas serão percorridas com a variável *diagonal_pixels*



Com esse pensamento para a diagonal superior esquerda vamos ter os valores dos pixels obtidos como:

```
# Diagonal superior esquerda
img_border[linha_atual-pixel][tamanho_borda-pixel]                = img[pixel][pixel]
if diagonal_pixels >= 2:
    for line in range(1, diagonal_pixels):
        # Linhas para esquerda da diagonal superior esquerda
        img_border[linha_atual-pixel][tamanho_borda-pixel-line]         = img[pixel][pixel+line]

        # Linhas para cima da diagonal superior esquerda
        img_border[linha_atual-pixel-line][tamanho_borda-pixel]         = img[pixel+line][pixel]
```

Analogamente iremos fazer isso para cada uma das 3 diagonais restantes, nos cenários 3, 6 e 8.