Relatório Trabalho 4

MC920 – Introdução ao Processamento de Imagem Digital

Julianny Favinha Donda RA 156059

1. Solução

1.1. Setup inicial

Uma transformação geométrica consiste em duas operações básicas, uma transformação espacial que define a reorganização dos pixels sobre o plano da imagem e uma interpolação de intensidade que trata da atribuição dos níveis de cinza ou cores aos pixels da imagem transformada espacialmente. Uma transformação espacial entre é uma função de mapeamento que permite a correspondência entre pontos de duas imagens. [1]

Seja um ponto (x, y) na imagem original que sofreu uma transformação para uma nova posição (x', y') na imagem resultante. A transformação espacial entre esses pontos é realizada por **mapeamento indireto**, o que garante que cada pontos da imagem resultante está associado à um ponto da imagem original.

$$\begin{pmatrix} x \\ y \end{pmatrix} = T^{-1} . \begin{pmatrix} x' \\ y' \end{pmatrix}$$

onde T^{-1} é a matriz inversa de transformação.

A matriz de transformação é definida a partir da operação que deseja-se realizar: rotação ou escala.

Caso seja uma rotação, então a matriz de transformação T a ser utilizada será

$$T = \begin{pmatrix} \cos(\,angle) & -\sin(\,angle) \\ \sin(\,angle) & \cos(\,angle) \end{pmatrix}$$

Caso seja uma escala, então a matriz de transformação T a ser utilizada será

$$T = \begin{pmatrix} scale\ in\ width & 0 \\ 0 & scale\ in\ height \end{pmatrix}$$

1.2. Interpolação pelo vizinho mais próximo

Na interpolação pelo vizinho mais próximo, simplesmente é necessário arredondar a posição (x, y) de cada pixel da nova imagem. Tal arredondamento foi feito usando a função `round()` pré-definida na linguagem. Com a nova posição (x, y), é feita uma verificação de limites da imagem original: se a posição existe na imagem original, então a intensidade desse pixel é retornada; caso contrário, é retornado 255 (intensidade branca). Essa verificação é implementada na função `boundary()`. Logo, a intensidade do pixel na nova imagem é então obtido executando a instrução a seguir

$$new\ image(x', y') = boundary(original\ image, round(x), round(y))$$

1.3. Interpolação bilinear

Na interpolação bilinear, os 4 pixels mais próximos do pixel (x, y) são considerados. A posição dos 4 pixels mais próximos de (x, y) foram determinadas da seguinte maneira:

$$position1 = (floor(x), floor(y))$$
$$position2 = (floor(x), ceil(y))$$
$$position3 = (ceil(x), floor(y))$$
$$position4 = (ceil(x), ceil(y))$$

As funções `floor()` e `ceil()` são as funções de arredondamento piso e teto, respectivamente, pré-definidas na linguagem.

Para cada pixel, é verificado se a posição existe na imagem original, usando a mesma verificação de limites da imagem original.

A distância entre os dois pares de pixels é calculada da seguinte maneira:

$$dx = x - floor(x)$$
$$dy = y - floor(y)$$

A intensidade do pixel na nova imagem é então obtido executando a instrução a seguir

$$new\ image(x', y') = (1 - dx) \cdot (1 - dy) \cdot boundary(original\ image, position1) + \\ + dx \cdot (1 - dy) \cdot boundary(original\ image, position2) + \\ + (1 - dx) \cdot dy \cdot boundary(original\ image, position3) + \\ + dx \cdot dy \cdot boundary(original\ image, position4)$$

1.4. Interpolação bicúbica

Na interpolação bicúbica, os 16 pixels mais próximos do pixel (x, y) são considerados (vizinhança 4x4). A intensidade do pixel na nova imagem é então obtido executando a equação a seguir

$$new\ image(x', y') = \sum_{m=-1}^{2} \sum_{n=-1}^{2} boundary(original\ image, x + m, y + n) \cdot R(m - dx) \cdot R(dy - n)$$

sendo

$$R(s) = 1/6 \cdot \left[P(s + 2)^3 - 4 \cdot P(s + 1)^3 + 6 \cdot P(s)^3 - 4 \cdot P(s - 1)^3\right]$$

$$P(t) = \begin{cases} t, & t > 0 \\ 0, & t \leq 0 \end{cases}$$

As distâncias dx e dy são calculadas da mesma forma como feito na Interpolação Bilinear.

1.5. Interpolação por Polinômios de Lagrange

Na interpolação por Polinômios de Lagrange, os 16 pixels mais próximos do pixel (x, y) são considerados (vizinhança 4x4). A intensidade do pixel na nova imagem é então obtido executando a equação a seguir

$$new\ image(x', y') = \frac{-dy \cdot (dy - 1) \cdot (dy - 2) \cdot L(1)}{6} + \frac{(dy + 1) \cdot (dy - 1) \cdot (dy - 2) \cdot L(2)}{2} + \\ + \frac{-dy \cdot (dy + 1) \cdot (dy - 2) \cdot L(3)}{2} + \frac{dy \cdot (dy + 1) \cdot (dy - 1) \cdot L(4)}{6}$$

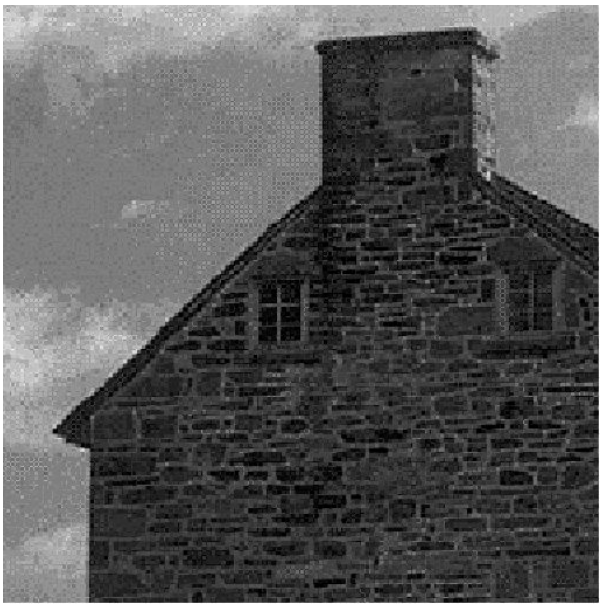
sendo

$$L(n) = \frac{-dx \cdot (dx - 1) \cdot (dx - 2) \cdot boundary(original\ image,\ x - 1,\ y + n - 2)}{6} +$$
$$+ \frac{(dx + 1) \cdot (dx - 1) \cdot (dx - 2) \cdot boundary(original\ image,\ x,\ y + n - 2)}{2} +$$
$$+ \frac{-dx \cdot (dx + 1) \cdot (dx - 2) \cdot boundary(original\ image,\ x + 1,\ y + n - 2)}{2} +$$
$$+ \frac{dx \cdot (dx + 1) \cdot (dx - 1) \cdot boundary(original\ image,\ x - 2,\ y + n - 2)}{6}$$

As distâncias dx e dy são calculadas da mesma forma como feito na Interpolação Bilinear.

2. Resultados

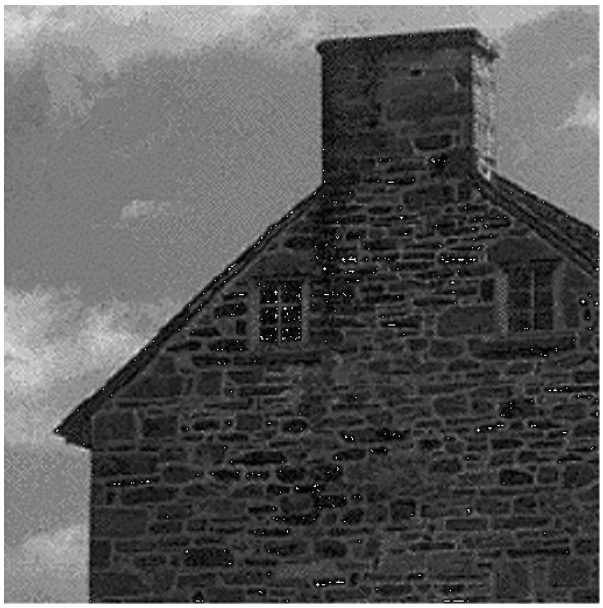
Figura - Imagem original, de dimensões 256 x 256 pixels.



(a) Interpolação pelo vizinho mais próximo.

(b) Interpolação bilinear.

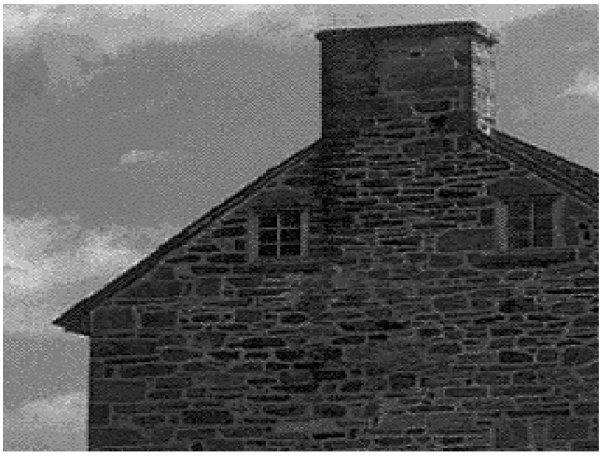
(c) Interpolação bicúbica.



(d) Interpolação por Polinômios de Lagrange.

Figura - Imagens resultantes para cada tipo de interpolação, após escala com fator 2.0. Dessa forma, as imagens resultantes possuem dimensões 512 x 512 pixels.

Nota-se que na Interpolação Bilinear, há um certo serrilhamento.



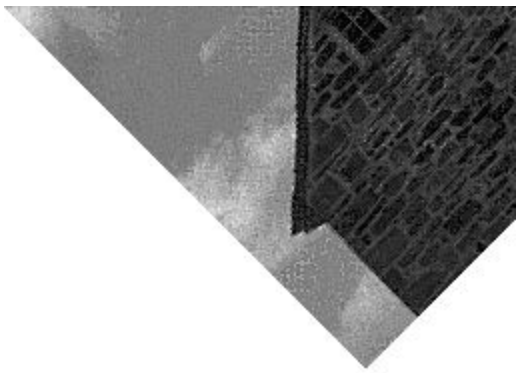
(a) Interpolação pelo vizinho mais próximo.

(b) Interpolação bilinear.

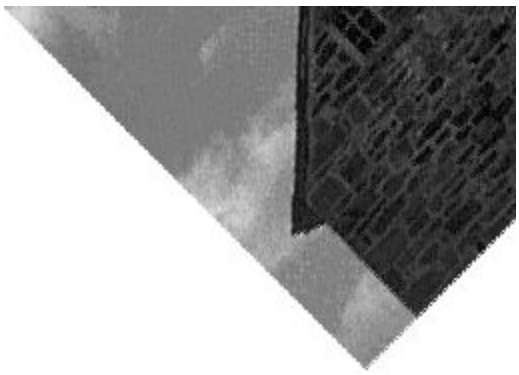
(c) Interpolação bicúbica.

(d) Interpolação por Polinômios de Lagrange.

Figura - Imagens resultantes para cada tipo de interpolação, após dimensão 1024 x 768 pixels. Note que a proporção da imagem resultante é diferente da proporção da imagem original, causando um esticamento na largura.

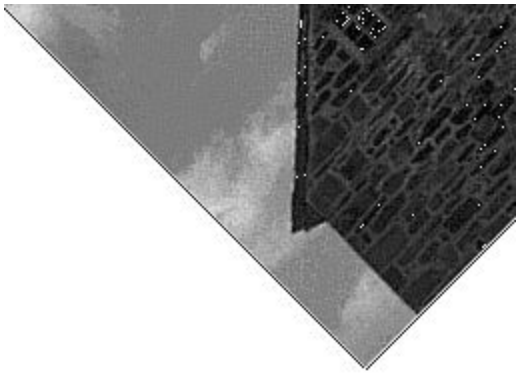


(a) Interpolação pelo vizinho mais próximo.



(b) Interpolação bilinear.

(c) Interpolação bicúbica.



(d) Interpolação por Polinômios de Lagrange.

Figura - Imagens resultantes para cada tipo de interpolação, após rotação de 45 graus em torno da origem.

A imagem foi rotacionada em torna da origem, ou seja, se aplicarmos uma rotação de 90 graus, a imagem não estará no campo visível da imagem original. Esse problema é resolvido se a imagem for rotacionada em torno de seu centroide.

3. Tempo de execução

Os tempos a seguir foram obtidos para uma imagem com dimensões 512 x 512 pixels.

Interpolação	Escala (2.0)	Dimensão (1024 x 768)	Rotação (50°)
Vizinho mais próximo	10.137184 s	18.340570 s	2.372077 s
Bilinear	13.181305 s	25.138001 s	3.155724 s
Bicúbica	436.358889 s	323.038124 s	323.038124 s
Polinômios de Lagrange	74.699346 s	75.243258 s	14.443854 s

Tabela 1 - Tempos de execução para diferentes operações e interpolações.

Nota-se que, apesar da Interpolação pelo Vizinho mais próximo apresentar a resposta mais rápida, perde-se um pouco da qualidade da imagem, como visto nas figuras anteriores. Em contrapartida, a Interpolação Bicúbica foi a mais demorada. As interpolações Bilinear e Bicúbica apresentaram um resultado satisfatório em relação à qualidade da imagem resultante.

4. Execução

É necessário que as bibliotecas **numpy** e **scikit-image** estejam instaladas. No terminal, execute

```
$ python3 geometric_transform.py [lista de parâmetros]
```

Para a `[lista de parâmetros]` existem três opções, listadas a seguir.

4.1. Rotação

Para rotacionar a imagem, é necessário informar o ângulo em graus.

```
$ python3 geometric_transform.py
            -a [ângulo em graus (ponto flutuante)]
            -m [modo]
            -i [nome da imagem de entrada].png
            -o [nome da imagem de saída].png
```

4.2. Escala

Para ampliar ou encolher a imagem, é possível informar o fator de escala. Note que para fatores entre 0.0 e 1.0 a imagem será encolhida, e para valores maiores que 1.0 a imagem será ampliada. O fator de escala tanto na altura quanto na largura da imagem são iguais.

```
$ python3 geometric_transform.py
            -e [escala em ponto flutuante]
            -m [modo]
            -i [nome da imagem de entrada].png
            -o [nome da imagem de saída].png
```

4.3. Dimensões

Para ampliar ou encolher a imagem, é possível informar também as dimensões (largura e altura) da imagem resultante. Esse é um caso mais geral de escala, pois como é informada a largura e a altura da imagem, é definido um fator de escala para para cada dimensão.

```
$ python3 geometric_transform.py
            -d [largura e altura da imagem de saída (inteiros)]
            -m [modo]
            -i [nome da imagem de entrada].png
            -o [nome da imagem de saída].png
```

Em todas as execuções, `modo` é a string “neighbor”, “bilinear”, "bicubic" ou "lagrange", indicando a técnica que deseja-se utilizar.

O programa `geometric_transform.py` gera o seguinte arquivo:

- Imagem `[nome da imagem de saída].png`, que é a imagem onde foi aplicada a técnica desejada.

Além disso, o programa exibe na saída padrão o tempo decorrido de execução.

5. Referências

[1] PEDRINI, Hélio. Registro de imagens. Disponível em <https://www.ic.unicamp.br/~helio/disciplinas/MC920/aula_registro.pdf>. p. 5-8, 14, 16-18. Acesso em 7 de junho de 2018.