Trabalho 5

Tiago de Paula Alves (187679)
tiagodepalves@gmail.com

31 de março de 2025

1 Introdução

Neste trabalho o objetivo foi implementar um ferramenta de transformações geométricas de escala e rotação em imagens. O resultado deve ser recuperado a partir de uma interpolação dos valores da imagem de entrada, após a transformação.

A ideia é comparar quatro métodos comuns de interpolação analisando seus resultados, mas considerando sua eficiência computacional. Os métodos serão comparados com três transformações distintas e imagens de dimensões variadas.

2 O Programa

A ferramenta foi desenvolvida e testada para as versões de Python[1] 3.6 ou superior. Também foram utilizados os pacotes OpenCV,[2] para entrada e saída de imagens, Numpy,[3] para operações vetorizadas com a imagem, e Matplotlib,[4] para opções de cores de fundo.

2.1 Código-Fonte

Neste trabalho foi elaborada a ferramenta `transforma.py` que faz a transformação da imagem e aplicação da interpolação. O código responsável pelas operações podem ser encontrados na pasta `lib`, como apresentado a seguir.

transforma.py Ferramenta de transformação da imagem.

lib Conjunto de arquivos com implementações para cada funcionalidade da ferramenta.

- **idx.py** Criação, transformação e acesso com as coordenadas de cada pixel da imagem.
- **linop.py** Operações lineares puras de escalonamento, rotação e translação.
- **opimg.py** Operações lineares mais aplicadas a imagem, sempre ajustando o resultado para a origem e retornando os novos limites da imagem.
- **interp.py** Recuperação da imagem transformada por interpolação dos pontos.
- **args.py** Processamento dos argumentos da linha de comando.
- **inout.py** Tratamento de entrada e saída de imagens do programa.
- **tipos.py** Tipos para checagem estática[5].[6]

Todas as imagens base para o processamento discutido ao longo do texto estão presente na pasta `imagens`. Também existe o *script* `run.sh` em Bash que refaz todos resultados apresentados neste relatório.

[1] *Python*. URL: `https://www.python.org/`.
[2] *OpenCV – Open Source Computer Vision Library*. URL: `https://opencv.org/`.
[3] *NumPy – Multidimensional Numerical Library for Python*. URL: `https://numpy.org/`.
[4] *Matplotlib: Python plotting*. URL: `https://matplotlib.org/`.
[5] *mypy: Optional Static Typing for Python*. URL: `http://mypy-lang.org/`.
[6] Guido van Rossum, Jukka Lehtosalo e Łukasz Langa. *PEP484 – Type Hints*. Python.org. Set. de 2014. URL: `https://www.python.org/dev/peps/pep-0484/`.

2.2 Argumentos

A execução de ambos os programas deverá ser feita através do interpretador de Python 3.6+. Os exemplos de execuções neste relatório podem não funcionar nessa versão, devido à ordem com que os argumentos são interpretados.[7] No entanto, o *script* `run.sh` funciona em todas variantes especificadas.

Todas as opções à seguir são apresentadas no texto de ajuda da ferramenta, que pode ser acessado com a *flag* `--help` ou apenas `-h`. É possível habilitar a amostragem do tempo de execução, com a opção `-v` ou `--verboso`. Quando repetida, essa opção apresenta ainda mais detalhes da execução da ferramenta.

2.2.1 Entrada e Saída

O único argumento obrigatório é o caminho da imagem de entrada, preferencialmente PNG, que deverá ser transformada. A imagem de saída é, por padrão, exibida em uma nova janela gráfica, mas pode ser salva em um arquivo com a opção `--output SAIDA` ou `-o SAIDA`. Tanto a entrada quanto a saída são tratadas em RGBA.[8]

Se nenhum outro argumento for especificado, a ferramenta apenas clona a imagem de entrada. Por exemplo, a invocação a seguir gera uma imagem `saida.png` idêntica à entrada `imagens/baboon.png`.

```
$ python3 transforma.py imagens/baboon.png -o saida.png
```

2.2.2 Transformações

Figura 1: Exemplo de execução.

A primeira transformação que pode ser feita na imagem é uma rotação no plano XY por um ângulo α. Isso pode ser realizado com `--angulo ALFA` ou `-a ALFA`. Também pode ser feita uma rotação em torno do eixo Y, que é projetada em perspectiva para o plano original da imagem. A *flag* para isso é `--beta BETA` ou `-b BETA`. Ambas opções tratam apenas de ângulos em graus. Os detalhes dessas transformações pode ser encontrado na seção 3.

Também existem as opções de escalonamento da imagem. A `--escala ESCALA` ou `-e ESCALA` recebe um fator de escala positivo e redimensiona cada pixel por esse fator, nos dois eixos da imagem. Também podem ser especificadas as dimensões da imagem resultante, com a *flag* `--dim ALTURA LARGURA` ou `-d ALTURA LARGURA`. Apenas uma das opções de escala pode ser usada a cada invocação.

As transformações são aplicadas na ordem com que aparecem nessa seção, ou seja, primeiro é feito a rotação no plano XY, depois a rotação em torno de Y com a projeção perspectiva, e por fim o redimensionamento. As entradas númericas também podem ser passadas por expressões matemáticas simples e podem usar funções da bibliotecas `math`. O exemplo abaixo usa algumas expressões para gerar a figura 1.

```
$ python3 transforma.py imagens/city.png -o saida.png \
    -a asin\(0.25\) -b deg\(pi/4\) -e 1/3
```

2.2.3 Método de Interpolação

A ferramenta é capaz de interpolar de quatro métodos diferentes, escolhidos com a opção `--metodo METODO` ou `-m METODO`. Os métodos são: `-m vizinho` (seção 4.1), para interpolação

[7] *argparse – Intermixed parsing*. Python. URL: https://docs.python.org/3/library/argparse.html#intermixed-parsing.

[8] *PNG Specification – Alpha channel*. W3C. URL: https://www.w3.org/TR/PNG-DataRep.html#DR.Alpha-channel.

pelo vizinho mais próximo, `-m bilinear` (4.2), `-m bicubica` (4.3) e `-m lagrange` (4.4), que usa polinômios de Langrange. O padrão é `bilinear`.

2.2.4 Cor do Fundo

Figura 2: Exemplo de execução.

Todas as transformações são feitas de forma que a imagem resultante seja uma caixa delimitadora da imagem transformada. No entanto, as rotações podem fazer com que a imagem transformada não seja retangular, o que deixa buracos na imagem resultante. Nesses casos, é aplicada uma cor de fundo em RGBA, representando a falta de informação.

Por padrão, a cor de fundo é transparente, mas isso pode ser alterado com a *flag* `--cor COR` ou `-c COR`. As opções de cores são tratadas pela função `to_rgba`,[9] então todas as cores da biblioteca Matplotlib são reconhecidas pela ferramenta. Uma opção extra `'transparente'` ou `'t'` é utilizada para representar o fundo transparente, usado como padrão.

O exemplo a seguir gera uma imagem figura 2, que é similar à 1, mas com fundo vermelho e construída por interpolação bicúbica.

```
$ python3 transforma.py imagens/city.png -o saida.png \
    -a asin\(0.25\) -b deg\(pi/4\) -e 1/3 -m bicubica -c red
```

3 Transformações

As transformações de imagem foram feitas por operações lineares em coordenadas homogêneas dos pixels. Inicialmente, as operações $\mathbf{T}$ são aplicadas nos pontos extremos $p_1 = (0,0,1)$, $p_2 = (0,H,1)$, $p_3 = (W,0,1)$ e $p_4 = (W,H,1)$ da imagem f, $H \times W$, de forma que os resultados $p'_i = T \cdot p_i$ pode ser usado para extrair os extremos da imagem transformada. Note que isso é válido, pois as transformações deste trabalho são colineações, isto é, elas mantém a colinearidade entre pontos. Assim, a caixa delimitadora[10] será dada por $(x_{\min}, y_{\min})$ e $(x_{\max}, y_{\max})$, sendo $x_{\min} = \min \left\{ \frac{x_i}{w_i} \;\middle|\; p'_i = (x_i, y_i, w_i) \right\}$ e assim por diante.

Usando a caixa delimitadora como a imagem resultante g, $H' \times W'$, temos os índices $y_{\min} \leq i \leq y_{\max}$ e $x_{\min} \leq j \leq x_{\max}$ como parte da imagem. Assim, podemos aplicar a operação inversa T' para descobrir qual o ponto equivalente na imagem original f. A última etapa é a interpolação com os valores discretos da imagem, discutida na seção 4 a seguir.

É importante notar, entretanto, que os pixels foram considerados pelo centro. Então, o pixel ij é entendido como $z_{ij} = f(j + 1/2, i + 1/2)$. Para efeito prático, isso feito por uma translação de $T_x = T_y = 1/2$ antes da transformação e outra $T_x = T_y = -1/2$ ao final. Isso faz com que as operações tenham o comportamento esperado.

A seguir estão apresentadas as transformações que podem ser aplicadas com a ferramenta. As matrizes foram baseadas na bibliografia da disciplina,[11] com algumas modificações para que a caixa delimitadora sempre comece na origem do plano cartesiano, ou seja, $x_{\min} = y_{\min} = 0$.

Além disso, as coordenadas homogêneas C foram representadas no código-fonte por um tensor $3 \times H \times W$ em que $C_{1ij} = x_{ij} = j$, $C_{2ij} = y_{ij} = i$ e $C_{3ij} = w_{ij} = 1$. Assim, a aplicação da

[9] *matplotlib.colors.to_rbga*. Matplotlib. URL: `https://matplotlib.org/api/_as_gen/matplotlib.colors.to_rgba.html` (acesso em 17/01/2021).

[10] *Minimum bounding box*. Wikipedia. URL: `https://en.wikipedia.org/wiki/Minimum_bounding_box`.

[11] Hélio Pedrini e William R. Schwartz. *Análise de imagens digitais: princípios, algoritmos e aplicações*. Cengage Learning Brasil, 2007. ISBN: 978-85-221-2836-5.

transformação **T** é feita pelo produto interno:

$$C'_{ijl} = \sum_{k=1}^{3} \mathbf{T}_{ik} C_{kjl}$$

3.1 Rotação no Plano XY

Esse tipo de rotação (em torno do eixo Z, perpendicular à imagem) pode ser feita por:

$$\mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

No programa, o ângulo α é tratado pelo oposto $-\alpha$, já que o eixo Y é invertido entre as representações matricial e cartesiana. Além disso, a rotação é combinada com uma translação **L** que corrige a posição da caixa delimitadora, sendo:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & -x_{\min} \\ 0 & 1 & -y_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

3.2 Escalonamento

Sendo S_x e S_y as escalas em X e Y, respectivamente, a matriz de mudança de escala é dada por:

$$\mathbf{S} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Para a opção de escala da entrada padrão, temos que $S_x = S_y$. No entanto, para o redimensionamento de $H_i \times W_i$ para $H_f \times W_f$, podemos usar $S_x = W_f/W_i$ e $S_y = H_f/H_i$. Note que operações de escala não precisam de translações corretivas.

Além disso, uma última operação de escala é feita após a transformação principal. Ela serve para arredondar as dimensões da imagem resultante $H \times W$ para os inteiros positivos mais próximos $H' \times W'$. Desse modo, $H' \geq 1$ e $H' - H \leq 1$, com restrições similares para W'.

3.3 Rotação em Torno do Eixo Y

Nessa transformação, a imagem é considerada na posição $Z = 0$ e é então rotacionada em torno de Y. O resultado disso é então projetado novamente no plano da imagem. Para isso, é preciso gerar o novo eixo, que é feito com a matriz **G**, para podermos rotacionar por β com **R**. Assim:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{R} = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A projeção perspectiva **P** é feita considerando o foco em $f = -1$ e a imagem em $Z = D$, que deve ser transladada com **D** para essa posição.

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & D \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{1}{f} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Além disso, temos as correções de escala. A primeira é uma correção da escala da projeção **E** com fatores $S_x = S_y = D - f = D + 1$. Isso vale, pois projeçoes são homografias, que preservam razões cruzadas. Temos também uma normalização **N**, resposável por reduzir a imagem

$H \times W$ para um quadrado 1×1, para podermos usar $D = 2$ como um padrão razoável. A normalização também faz uma translação de $T_x = T_y = -1/2$ para centralizar a imagem na origem.

$$\mathbf{E} = \begin{bmatrix} D+1 & 0 & 0 \\ 0 & D+1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{N} = \begin{bmatrix} 1 & 0 & -\frac{1}{2} \\ 0 & 1 & -\frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{W} & 0 & 0 \\ 0 & \frac{1}{H} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Assim, a transformação final é dada $\mathbf{B} = \mathbf{N}^{-1} \cdot \mathbf{E} \cdot \mathbf{P} \cdot \mathbf{D} \cdot \mathbf{R} \cdot \mathbf{G} \cdot \mathbf{N}$. Nesse caso, assim como na rotação anterior (seção 3.3), é necessária a correção da posição da caixa delimitadora, feito com uma matriz similar à **L** naquela situação.

4 Métodos de Interpolação

Após a etapa descrita na seção anterior, teremos a transformação **T** e as dimensões $H_g \times W_g$ da imagem de saída. Assim, para cada ponto (x_g, f_g), podemos encontrar o ponto da entrada (x_f, y_f) associado:

$$\begin{bmatrix} x_f \\ y_f \\ 1 \end{bmatrix} = \begin{bmatrix} wx_f \\ wy_f \\ w \end{bmatrix} = \mathbf{T}^{-1} \cdot \begin{bmatrix} x_g \\ y_g \\ 1 \end{bmatrix}$$

Com isso, temos $g(x_g, y_g) = f(x_f, y_f)$ descrevendo a intensidade da imagem resultante. No entanto, como a imagem é discreta, f só é definida em pontos específicos, no caso, em inteiros dentro dos limites da imagem. Para resolver isso, podemos usar uma função interpolada f' que aproximam o valor com $f'(x_f, y_f)$, usando a vizinhança de (x_f, y_f). Assim, com base na figura 3, nós temos $x = \lfloor x_f \rfloor$ e $y = \lfloor y_f \rfloor$, além de $dx = x_f - x$ e $dy = y_f - y$, que são os valores que definem vizinhança e as distâncias para interpolação.

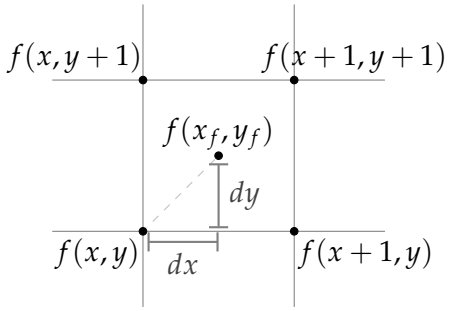


Figura 3: Vizinhança de (x_f, y_f).

Para pontos fora da figura, podemos usar a cor de fundo (escolhida como na seção 2.2.4) como valor do pixel. Assim, $f(x, y) = z_{\text{fundo}}$ quando $x < 0$, $x \geq W_f$, $y < 0$ ou $y \geq H_f$.

4.1 Pelo Vizinho Mais Próximo

Nesse método, a intensidade da pixel é escolhido pelo vizinho com posição mais próxima de (x_f, y_f). Assim:

$$\begin{aligned} g(x_g, y_g) = f'(x_f, y_f) &= \begin{cases} f(x, y) & \text{se } dx < 0.5 \text{ e } dy < 0.5 \\ f(x+1, y) & \text{se } dx \geq 0.5 \text{ e } dy < 0.5 \\ f(x, y+1) & \text{se } dx < 0.5 \text{ e } dy \geq 0.5 \\ f(x+1, y+1) & \text{se } dx \geq 0.5 \text{ e } dy \geq 0.5 \end{cases} \\ &= f(\text{round}(x_f), \text{round}(y_f)) \end{aligned}$$

4.2 Bilinear

Na interpolação bilinear, a intensidade é interpolada por uma *spline*[12] linear na dimensão X e outra *spline* em Y. O resultado deverá seguir a seguinte fórmula:

$$\begin{aligned} f'(x_f, y_f) &= (1-dx)\left[(1-dy)f(x,y) + dyf(x,y+1)\right] \\ &\quad + dx\left[(1-dy)f(x+1,y) + dyf(x+1,y+1)\right] \\ &= (1-dx)(1-dy)f(x,y) + dx(1-dy)f(x+1,y) \\ &\quad + (1-dx)dyf(x,y+1) + dxdyf(x+1,y+1) \end{aligned}$$

4.3 Bicúbica

Assim como na interpolação bilinear, a bicúbica é aproximada por *splines* na malha retangular. No caso, é interpolação é feita por B-*splines*[13] de ordem 3, que mantém continuidade na primeira derivada,[14] apesar de serem descritas por partes. Na malha retangular, essa aproximação é dada por:

$$f'(x_f, y_f) = \sum_{m=-1}^{2} \sum_{n=-1}^{2} R(m-dx)R(dy-n) \cdot f(x+m, y+n)$$

Sendo que:

$$\begin{aligned} R(s) &= \frac{1}{6}\left[P(s+2)^3 - 4P(s+1)^3 + 6P(s)^3 - 4P(s-1)^3\right] \\ P(t) &= \max\{t, 0\} = \begin{cases} t, & \text{se } t > 0 \\ 0, & \text{caso contrário} \end{cases} \end{aligned}$$

4.4 Por Polinômios de Lagrange

Nesse último método, também é feita uma interpolação cúbica nas dimensões X e Y separadamente, por isso ele também pode ser chamado de interpolação bicúbica, como no método anterior. Aqui, no entanto, a função encontrada é um polinômio propriamente, sem funções por partes, comumente denominado polinômio de Lagrange.[15] A fórmula do método é:

$$\begin{aligned} f'(x_f, y_f) &= \frac{-dy(dy-1)(dy-2)}{6} \cdot L(1) + \frac{(dy+1)(dy-1)(dy-2)}{2} \cdot L(2) \\ &\quad + \frac{-dy(dy+1)(dy-2)}{6} \cdot L(3) + \frac{dy(dy+1)(dy-1)}{2} \cdot L(4) \end{aligned}$$

Considerando:

[12] *Spline interpolation*. Wikipedia. URL: `https://en.wikipedia.org/wiki/Spline_interpolation`.

[13] Eric W. Weisstein. "B-Spline". Em: *MathWorld – A Wolfram Web Resource* (2000). URL: `https://mathworld.wolfram.com/B-Spline.html`.

[14] Quentin Agrapart e Alain Batailly. *Cubic and bicubic spline interpolation in Python*. Rel. técn. École Polytechnique de Montréal, 2020. URL: `https://hal.archives-ouvertes.fr/hal-03017566`.

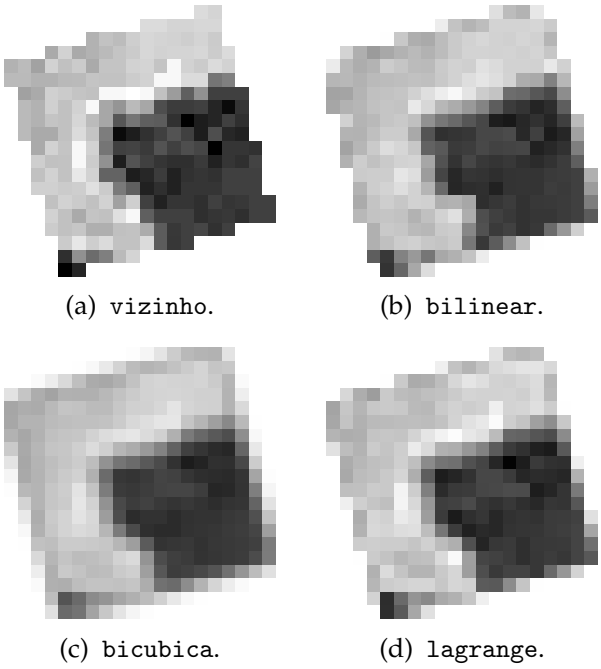
[15] Eric W. Weisstein e Branden Archer. "Lagrange Interpolating Polynomial". Em: *MathWorld – A Wolfram Web Resource* (2000). URL: `https://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html`.

$$
\begin{aligned}
L(n) = & \frac{-dx(dx-1)(dx-2)}{6} \cdot f(x-1, y+n-2) \\
& + \frac{(dx+1)(dx-1)(dx-2)}{2} \cdot f(x, y+n-2) \\
& + \frac{-dx(dx+1)(dx-2)}{6} \cdot f(x+1, y+n-2) \\
& + \frac{dx(dx+1)(dx-1)}{2} \cdot f(x+2, y+n-2)
\end{aligned}
$$

Uma desvatagem desse método é que as derivadas de primeira ordem normalmente são descontínuas, que pode causar efeitos de descontinuidade visual na imagem resultante. No entanto, isso também ajuda a evitar efeitos de *blur* não intencionais.[16]

5 Resultados

5.1 Rotações



(a) `vizinho`. (b) `bilinear`.

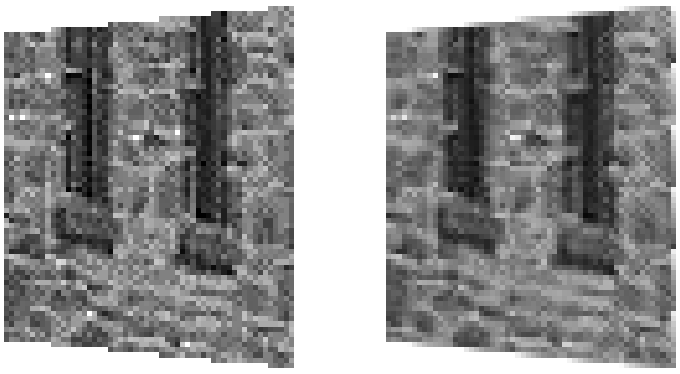
(c) `bicubica`. (d) `lagrange`.

Figura 4: Rotação de 15° no plano da imagem aplicada em `house16.png` (16×16).

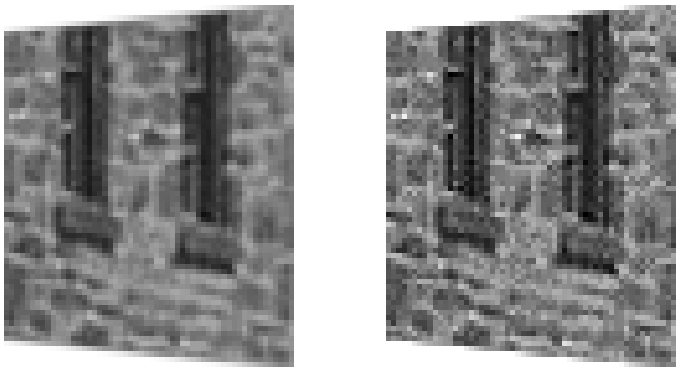
Na figura 5, podemos ver com clareza a diferença entre os métodos. Nas interpolações bilinear e bicúbica, a imagem resultante aparece com um pequeno borramento. Esse efeito é bem mais fraco com polinômios de Lagrange.

Para o aproximação por vizinho mais próximo, a figura fica com um serrilhado bem presente, principalmente nas bordas. Isso aparece inclusive em imagens grandes, como na figura 6. Além desse método, quase não existem diferenças visuais para as imagens de 512×512.

[16] Gwanggil Jeon. "Lagrange Interpolation for Upsampling". Em: *International Journal of Multimedia and Ubiquitous Engineering* 10.7 (2015). DOI: 10.14257/ijmue.2015.10.7.35.



(a) `vizinho`. (b) `bilinear`.



(c) `bicubica`. (d) `lagrange`.

Figura 5: Rotação de -30° em torno do eixo Y aplicada em `house64.png` (64×64).

(a) `vizinho`. (b) `bilinear`.

(c) `bicubica`. (d) `lagrange`.

Figura 6: Rotação de 15° no plano da imagem aplicada em `house.png` (512×512).

5.2 Escalonamento

(a) `vizinho`. (b) `bilinear`.

(c) `bicubica`. (d) `lagrange`.

Figura 7: Escalonamento com $S_x = S_y = 1/3$ aplicado em `city.png` (512×512).

(a) `vizinho`. (b) `bilinear`.

(c) `bicubica`. (d) `lagrange`.

Figura 8: Escalonamento com $S_x = S_y = 1.5$ aplicado em `city128.png` (128×128).

No caso da mudança de escala, podemos ver que a bicúbica tem os melhores resultados na redução das dimensões, que acontece na figura 7. No entanto, para ampliação da imagem (figura 8), o borramento da imagem é muito forte, fazendo com que o método por polinômios

de Lagrange tenha resultados melhores. Isso também acontece na figura 9, por ser muito pequeno o resultado.

Tanto na ampliação (8b), quanto na redução (7b), o método bilinear gerou artefatos muito presentes nas bordas dos objetos. No entanto, isso pode ser devido à implementação e não necessariamente ao método. Um efeito similar aconteceu usando os vizinhos mais próximos.

(a) `vizinho`.

(b) `bilinear`.

(c) `bicubica`.

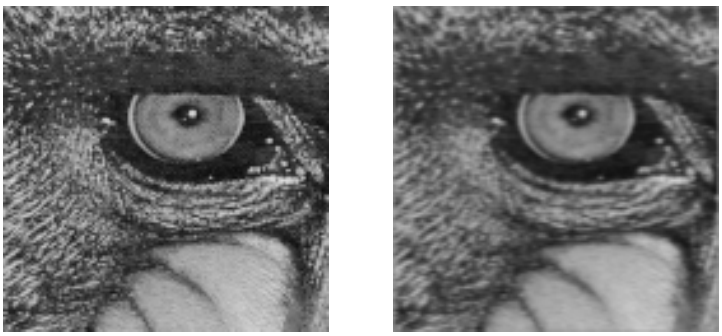
(d) `lagrange`.

Figura 9: Redimensionamento para 80×60 aplicado em `city128.png` (128×128).

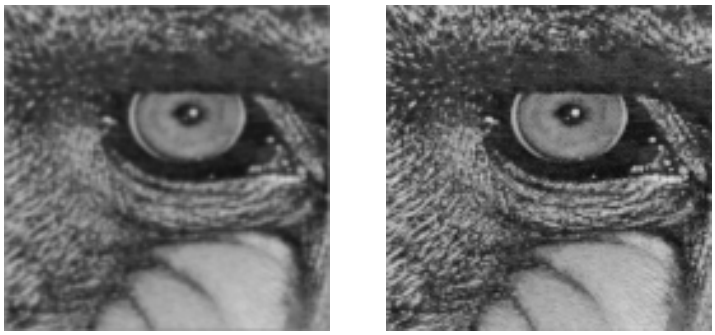
5.3 Reconstrução

Figura 10: `baboon128.png` (128×128).

Tentando aplicar uma transformação simples, de escala (figura 11) ou de rotação (figura 12), seguida da sua inversa podemos ver que as interpolações bilinear e bicúbica causam um efeito de *blur* muito presente. Para valores racionais, como $S_x = S_y = 1.37$, pode ser que o arredondamento acabe causando artefatos maiores no método dos vizinhos mais próximos.

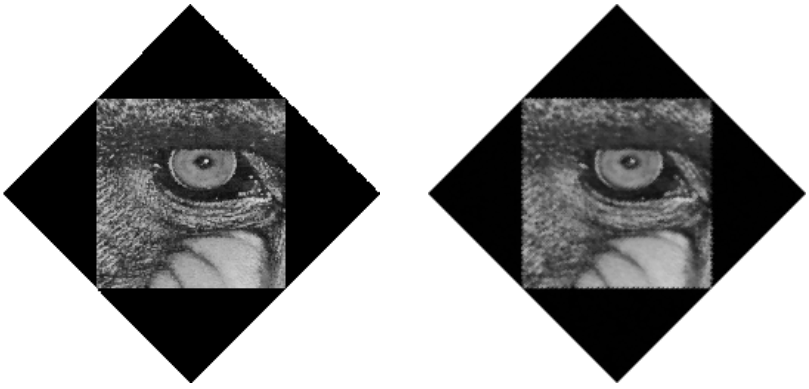


(a) `vizinho`. (b) `bilinear`.

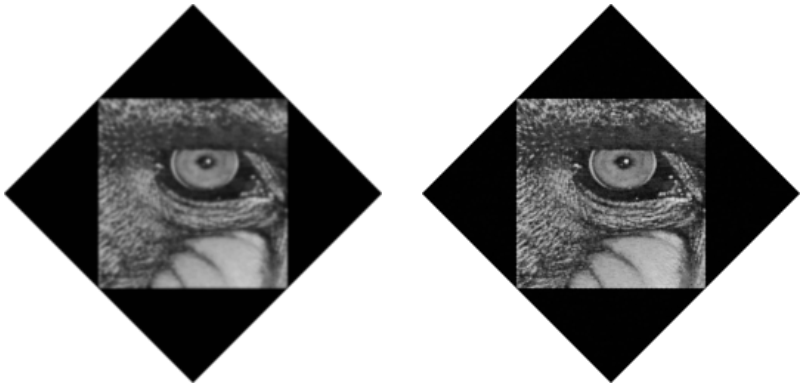


(c) `bicubica`. (d) `lagrange`.

Figura 11: Escalonamento por um fator 2 seguido de outro escalonamento de fator 1/2 em `baboon128.png`, usando o mesmo método em cada caso.



(a) `vizinho`. (b) `bilinear`.



(c) `bicubica`. (d) `lagrange`.

Figura 12: Rotação de 45° (com borda preta) seguida de outra rotação de -45° no plano da imagem em `baboon128.png`, usando o mesmo método em cada caso.

5.4 Tempo de Execução

Figura 13: `among.png` (2000 × 1544).

Para testar a eficiência computacional de cada método, a imagem `among.png` (13) foi rotacionada e escalonada por 2, gerando uma imagem final de dimensões 4867 × 4112 pixels em RGBA, ou seja, com 4 canais de cores. O tempo de execução pode ser medido na própria ferramenta, no modo verboso `-v`, como exemplificado abaixo para o método `lagrange`.

```
$ python3 transforma.py imagens/among.png -o saida.png -v \
    -c r -a 22 -b 20 -e 2 -m lagrange
INFO:root:imagem imagens/among.png de dimensões (2000, 1544, 4)
INFO:root:transformação em 0.34468913078308105 segundos
INFO:root:interpolação em 23.732824563980103 segundos
```

Cada método também foi executado com um dos *profilers* padrões de Python, o `cProfile`.[17] O profiler é capaz de mostrar o tempo de cada função separadamente e cumulativamente. Por exemplo, uma execução do método `bicubica`, poderia ter o seguinte resultado:

```
$ python3 -m cProfile -s cumtime \
    transforma.py imagens/among.png -o saida.png \
    -c r -a 22 -b 20 -e 2 -m bicubica
    149578 function calls (145313 primitive calls) in 46.493 seconds

    Ordered by: cumulative time

    ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     467/1    0.001    0.000   46.493   46.493 {built-in method builtins.exec}
         1    0.000    0.000   46.493   46.493 transforma.py:1(<module>)
         1    0.020    0.020   45.205   45.205 interp.py:23(__call__)
         1    9.408    9.408   45.185   45.185 interp.py:125(bicubica)
        32    8.006    0.250   23.462    0.733 interp.py:138(R)
       128   15.454    0.121   15.455    0.121 interp.py:130(Pe3)
        16   11.768    0.735   11.768    0.736 idx.py:75(acesso)
         1    0.000    0.000    0.685    0.685 inout.py:77(imgwrite)
...
```

[17] *The Python Profilers – Instant User's Manual*. Python. URL: `https://docs.python.org/3/library/profile.html#instant-user-s-manual`.

Tabela 1: Tempos de execução dos métodos e das funções principais.

	Parte ou Função	`vizinho`	`bilinear`	`bicubica`	`lagrange`
Opção `-v`	Transformação	351 ms	347 ms	348 ms	346 ms
	Interpolação	0.96 s	4.84 s	46.27 s	23.45 s
`cProfile`	`transforma.py:(transformacao)`	0.351 s	0.347 s	0.343 s	0.345 s
	`interp.py:(__call__)`	0.950 s	4.762 s	45.963 s	23.527 s
	`idx.py:(acesso)`	0.801 s	3.087 s	12.440 s	12.062 s
	`interp.py:(modf)`	—	0.237 s	0.236 s	0.238 s
	`interp.py:(asimg)`	—	0.183 s	0.305 s	0.288 s
	`interp.py:(R)`	—	—	23.578 s	—
	`interp.py:(Pe3)`	—	—	15.538 s	—
	`interp.py:(L)`	—	—	—	20.921 s

Os resultados de uma execução estão na tabela 1. Por mais que foi apenas uma execução, outros testes obtiveram resultados similares, oscilando em menos de 20%. Como os valores são bem distintos, isso é o bastante para a análise.

6 Conclusão

Podemos ver que a diferenças dos métodos de interpolação é grande, tanto em relação ao resultado quanto ao tempo necessário para a aplicação. Os métodos mais custosos são os que tem melhores resultados visuais, no caso as interpolações cúbica e por polinômios de Lagrange. No entanto, o método de Lagrange consegue ser melhor em várias situações, por evitar borramento da imagem, mesmo sendo bem mais eficiente.

Em algumas situações, como no caso de redução das dimensões da imagem, a interpolação bicúbica pode ser melhor. O método pode ser melhorado ainda mais com algum filtro de frequência, como um de nitidez. Em aplicações como redes neurais, em que *downsampling* é muito importante, esse método pode ser bem útil.

Em outras situações em que *downsampling* também pe necessário, mas não afeta tanto o resultado, a interpolação por vizinho mais próximo pode ser mais interessante. Isso por sua eficiência, capaz de produzir imagens com 20 mega pixels em poucos segundos.