

Relatório EP2 - MAC 0210

Julia Leite e Lara Ayumi Nagamatsu

01 de junho de 2020

Resumo

Este Exercício Programa (EP), proposto na matéria de Laboratório de Métodos Numéricos (MAC 0210), envolve comprimir uma imagem e depois reconstruí-la, utilizando dois métodos de interpolação por polinômios, a *interpolação bilinear por partes* e a *interpolação bicúbica*, para o caso bivariado.

Sumário

1	Introdução	2
2	Participantes	2
3	Arquivos	2
3.1	compress.m	2
3.2	decompress.m	2
3.2.1	Interpolação Bilinear	3
3.2.2	Interpolação Bicúbica	3
3.3	calculateError.m	4
4	Experimentos	5
4.1	Parte 1: Zoológico	5
4.1.1	Exemplos de imagens em preto e branco	5
4.1.2	Exemplos de Imagens coloridas	7
4.1.3	Conclusões	8
4.2	Parte 2: Selva	9
4.2.1	Exemplos de imagens	9
4.2.2	Conclusões	11

1 Introdução

Neste EP, comprimimos uma imagem e a reconstruímos utilizando dois métodos de interpolação por partes: a *Interpolação Bilinear* e a *Interpolação Bicúbica*. O objetivo reside em observar as diferenças nessas duas formas de interpolação.

Podemos, então, visualizar como esses métodos se comportam em dois tipos de situação: quando a imagem é gerada por uma função (*Zoológico*) ou em casos reais (*Selva*).

2 Participantes

- NUSP:11221797 - Julia Leite
- NUSP: 9910568 - Lara Ayumi Nagamatsu

3 Arquivos

Este programa é composto de 3 *functions files* do Octave: **compress.m**, onde ocorre a compressão da imagem, **decompress.m**, função na qual descomprimimos a imagem de acordo com o método escolhido e **calculateError.m**, método utilizado para calcular o erro.

3.1 compress.m

Neste arquivo recebemos o nome da imagem *originalImg* e a taxa de compressão *k* descritos no enunciado do EP e utilizamos, então, o *imread*, para carregarmos a imagem e seu colormap (para o caso de a imagem não ser RGB).

Verificamos, então, se a imagem é quadrada. Se não o for, recortamos parte dela para torná-la quadrada. Neste caso, por exemplo, se originalmente a imagem tivesse tamanho *m*x*n*, com *m* < *n*, a imagem se torna *m*x*m*. Conferimos, também, se a imagem é RGB, já que isso determina o procedimento adotado pelo programa.

Calculamos, em seguida, o tamanho da imagem comprimida, *n*, de acordo com a fórmula:

$$n = \frac{img_p + k}{k+1} , \text{ sendo } img_p \text{ o número de pixels da imagem original.}$$

Por padrão, pegamos o *floor* de *n*, contudo, em algum casos, isso fazia com o que o número de pixels da imagem reconstruída (*p*) fosse menor que o número de pixels da imagem original. Para evitar isso, calculamos o *p* e, se ele for menor que o número de pixels da imagem original, fazemos com que o *n* seja o *ceil* da expressão.

Podemos obter o número de pixels da imagem reconstruída através de: *p* = *n* + (*n* - 1)*k*.

Construída a imagem de tamanho *n*x*n*, finalmente a exibimos na tela, e a salvamos no arquivo "compressed.png".

3.2 decompress.m

Este arquivo recebe o nome da imagem comprimida (*compressedImg*), o método escolhido para descomprimi-la, *method*, sua taxa de compressão *k* e o *h* utilizado no cálculo das distâncias de pontos a serem gerados (dado pela distância entre dois pixels da imagem comprimida).

Primeiro, carregamos a imagem e seu colormap, além disso, obtemos sua altura, comprimento e dimensões, conferimos, então, se a imagem é quadrada, caso contrário, paramos a execução do programa. Calculamos, também, o tamanho (número de pixels) da altura da imagem, *p*.

Criamos então as matrizes das componentes RGB da nova imagem e atribuímos os valores dos pixels da imagem comprimida nas posições correspondentes na imagem descomprimida. Os pixels restantes recebem -1 como valor inicial para que seja possível, durante a interpolação, verificar se um pixel já foi preenchido.

Para reconstruirmos a imagem, associamos quadrados no R^2 a quadrados Q_{ij} de pixels, com os pixels dos vértices conhecidos, ou seja, pertencentes à imagem comprimida, e sendo (x_i, y_j) o ponto inferior esquerdo e (\bar{x}, \bar{y}) seu correspondente no quadrado dos números reais. O ponto superior direito, portanto, do quadrado Q_{ij} é (x_{p-1}, y_{p-1}) e seu correspondente no quadrado dos reais, $(\bar{x} + (p-1)h, \bar{y} + (p-1)h)$.

Dessa forma, o valor de um pixel (i, j) não pertencente à imagem comprimida pode ser obtido por

$$(i, j) = f(x_i, y_j) = f(\bar{x} + ih, \bar{y} + jh)$$

A função f é determinada pelo método de interpolação escolhido. Ao final da interpolação, salvamos a imagem como "decompress.png" e a exibimos na tela.

3.2.1 Interpolação Bilinear

Neste método, utilizamos a seguinte fórmula para determinar as cores, $P_{ij}(x, y)$, dos pixels interpolados:

$$f(x, y) = P_{ij}(x, y) = a_0 + a_1(x - x_i) + a_2(y - y_j) + a_3(x - x_i)(y - y_j)$$

Os coeficientes de $P_{ij}(x, y)$ são armazenados no vetor $coef$. Seus valores são determinados a partir dos cálculos:

$$\begin{aligned} coef(1) &= a_0 = P_{ij}(x_i, y_j) \\ coef(2) &= a_2 = \frac{P_{ij}(x_i, y_{j+1}) - a_0}{h} \\ coef(3) &= a_1 = \frac{P_{ij}(x_{i+1}, y_j) - a_0}{h} \\ coef(4) &= a_3 = \frac{P_{ij}(x_{i+1}, y_{j+1}) - a_0}{h^2} - \frac{a_1}{h} - \frac{a_2}{h} \end{aligned}$$

A cada iteração recalculamos os coeficientes e interpolamos os pixels de um quadrado de lado $(k+2)$ da imagem. Então, por exemplo, na primeira iteração desse método, interpolaremos os pontos do quadrado $[(1,1), (1, k+2), (k+2,1), (k+2, k+2)]$.

3.2.2 Interpolação Bicúbica

Nesse método, o valor de um pixel P_{ij} pertencente a um quadrado Q_{ij} é dado por:

$$P_{ij}(x, y) = [1 \quad (x - x_i) \quad (x - x_i)^2 \quad (x - x_i)^3] \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} 1 \\ (y - y_j) \\ (y - y_j)^2 \\ (y - y_j)^3 \end{bmatrix}$$

Precisamos, então encontrar os 16 coeficientes da matriz de coeficientes, $COEF$, para cada quadrado Q_{ij} e, para tal, utilizamos as seguintes matrizes:

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & h & h^2 & h^3 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2h & 3h^2 \end{bmatrix}$$

Chamamos de BT a transposta da matriz B , BI a inversa da matriz B , BTI a inversa da BT . Então, podemos determinar a matriz $COEF$ através da relação:

$$A = \begin{bmatrix} f(x_i, y_j) & f(x_i, y_{j+1}) & \frac{\partial f}{\partial y}(x_i, y_j) & \frac{\partial f}{\partial y}(x_i, y_{j+1}) \\ f(x_{i+1}, y_j) & f(x_{i+1}, y_{j+1}) & \frac{\partial f}{\partial y}(x_{i+1}, y_j) & \frac{\partial f}{\partial y}(x_{i+1}, y_{j+1}) \\ \frac{\partial f}{\partial x}(x_i, y_j) & \frac{\partial f}{\partial x}(x_i, y_{j+1}) & \frac{\partial^2 f}{\partial x \partial y}(x_i, y_j) & \frac{\partial^2 f}{\partial x \partial y}(x_i, y_{j+1}) \\ \frac{\partial f}{\partial x}(x_{i+1}, y_j) & \frac{\partial f}{\partial x}(x_{i+1}, y_{j+1}) & \frac{\partial^2 f}{\partial x \partial y}(x_{i+1}, y_j) & \frac{\partial^2 f}{\partial x \partial y}(x_{i+1}, y_{j+1}) \end{bmatrix} = B * COEF * BT$$

$$COEF = BI * \begin{bmatrix} f(x_i, y_j) & f(x_i, y_{j+1}) & \frac{\partial f}{\partial y}(x_i, y_j) & \frac{\partial f}{\partial y}(x_i, y_{j+1}) \\ f(x_{i+1}, y_j) & f(x_{i+1}, y_{j+1}) & \frac{\partial f}{\partial y}(x_{i+1}, y_j) & \frac{\partial f}{\partial y}(x_{i+1}, y_{j+1}) \\ \frac{\partial f}{\partial x}(x_i, y_j) & \frac{\partial f}{\partial x}(x_i, y_{j+1}) & \frac{\partial^2 f}{\partial x \partial y}(x_i, y_j) & \frac{\partial^2 f}{\partial x \partial y}(x_i, y_{j+1}) \\ \frac{\partial f}{\partial x}(x_{i+1}, y_j) & \frac{\partial f}{\partial x}(x_{i+1}, y_{j+1}) & \frac{\partial^2 f}{\partial x \partial y}(x_{i+1}, y_j) & \frac{\partial^2 f}{\partial x \partial y}(x_{i+1}, y_{j+1}) \end{bmatrix} * BTI$$

Para facilitar o cálculo dos elementos da matriz A , definimos:

$$deij = \frac{\partial f}{\partial y}(x_{i-1}, y_j) = \frac{f(x_{i-1}, y_{j+1}) - f(x_{i-1}, y_{j-1})}{2h}$$

$$deiJ = \frac{\partial f}{\partial y}(x_{i-1}, y_{j+1}) = \frac{f(x_{i-1}, y_{j+2}) - f(x_{i-1}, y_j)}{2h}$$

$$delIj = \frac{\partial f}{\partial y}(x_{i+2}, y_j) = \frac{f(x_{i+2}, y_{j+1}) - f(x_{i+2}, y_{j-1})}{2h}$$

$$delIJ = \frac{\partial f}{\partial y}(x_{i+2}, y_{j+1}) = \frac{f(x_{i+2}, y_{j+2}) - f(x_{i+2}, y_j)}{2h}$$

O cálculo das derivadas acima é modificado caso o ponto seja de borda, conforme descrito no enunciado do EP. Então, após calculamos os elementos da matriz A , determinamos os elementos da matriz $COEF$ e interpolamos os pontos de cada quadrado Q_{ij} da imagem.

3.3 calculateError.m

Neste arquivo recebemos o nome da imagem original, $originalImg$, e da imagem descomprimida, $decompressedImg$, carregamos as imagens e obtemos a altura, comprimento e dimensão de ambas.

Obtemos as matrizes das componentes RGB da imagem original ($origR, origG, origB$) e da imagem descomprimida, $decR, decG, decB$. Caso as imagens não tenham o mesmo tamanho, por conta da escolha de $floor$ ou $ceil$ do n, retiramos uma linha e uma coluna das matrizes da imagem maior até que ambas tenham o mesmo comprimento e, consequentemente, a mesma altura, já que ambas são quadradas. Imagens de 1 dimensão (não RGB) são tratadas similarmente: os cálculos são os mesmos, mas são feitos para apenas uma matriz, e não 3.

Dessa forma, calculamos o erro através das fórmulas:

$$errR = \frac{\|origR - decR\|_2}{\|origR\|_2}$$

$$err = \frac{errR + errG + errB}{3}$$

sendo os erros $errG$ e $errB$ calculados de maneira análoga.

4 Experimentos

4.1 Parte 1: Zoológico

As imagens geradas nesta seção foram construídas por meio de funções.

Observação: As seguintes imagens foram geradas com os parâmetros $k = 7$ e $h = 50$. Para cada função gerada, o intervalo dado para seu contradomínio é mapeado nos valores correspondentes, de 0 a 255. Neste caso, por exemplo:

$$aux = 255 * aux$$

$$aux = aux + 255$$

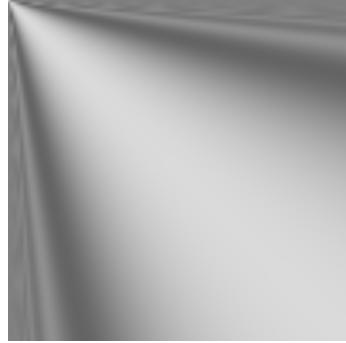
$$aux = aux/2$$

para $aux \in [-1,1]$, uma função.

4.1.1 Exemplos de imagens em preto e branco



(a) Original



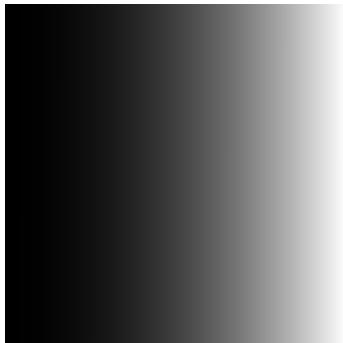
(b) Bilinear



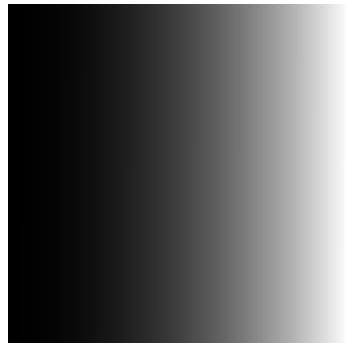
(c) Bicúbica

Figura 1: $f(x, y) = (\sin(\frac{x}{y})\sin(\frac{y}{x}), \sin(\frac{x}{y})\sin(\frac{y}{x}), \sin(\frac{x}{y})\sin(\frac{y}{x}))$

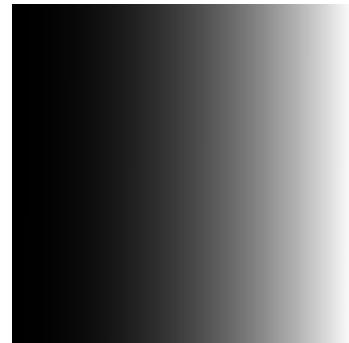
Na figura 1, o erro resultante foi 0.0037261 para a interpolação bilinear e 0.0036784 para a bicúbica.



(a) Original



(b) Bilinear



(c) Bicúbica

Figura 2: $f(x, y) = (3y^2 + x, 3y^2 + x, 3y^2 + x)$

Na figura 2, o erro foi 0.0036113 para a imagem bilinear e 0.0039840 para a imagem bicubica.

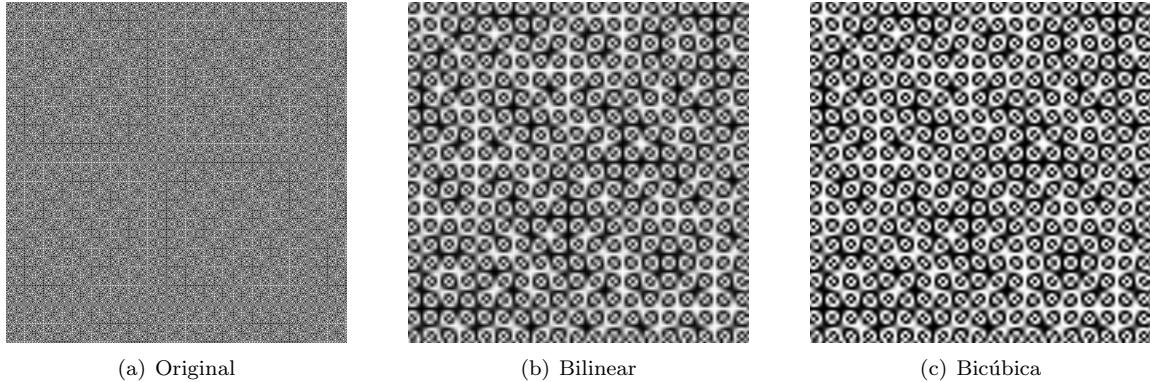


Figura 3: $f(x, y) = (\sin(xy), \sin(xy), \sin(xy))$

Na figura 3, o erro encontrado foi 0.091067 para a imagem bilinear e 0.09822 para a imagem bicubica.

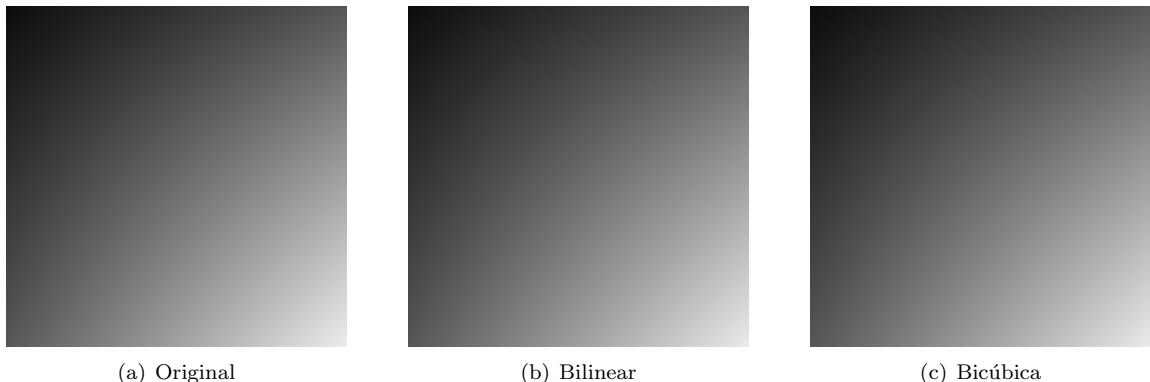


Figura 4: $f(x, y) = (xy, x + y, 100)$ em grayscale

Na figura 4, o erro encontrado foi $8.4433e^{-4}$ para a imagem bilinear e de $8.6519e^{-4}$ para a imagem bicubica.

4.1.2 Exemplos de Imagens coloridas

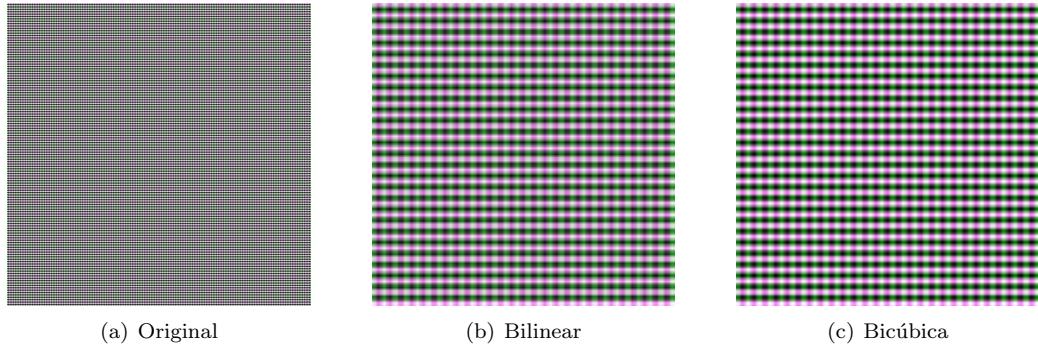


Figura 5: $f(x, y) = (\sin(x), \frac{\sin(x)+\sin(y)}{2}, \sin(x))$

A figura 5 representa a função cuja análise foi **requisitada** no EP. O erro encontrado para suas descompressões foi de 0.61176 para a imagem bilinear e de 0.66915 para a imagem bicubica.

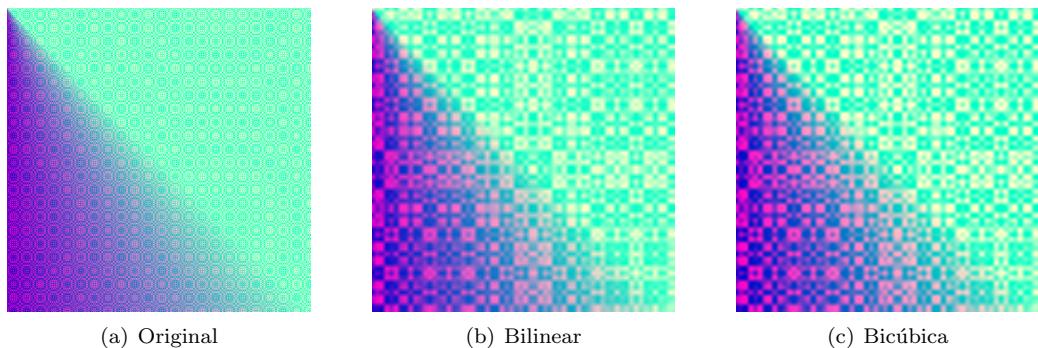


Figura 6: $f(x, y) = (\sin(x^2 + y^2), 200, \frac{y}{x})$

Na imagem 6, o erro foi 0.17204 para a imagem bicubica e 0.17207 para a imagem bilinear.

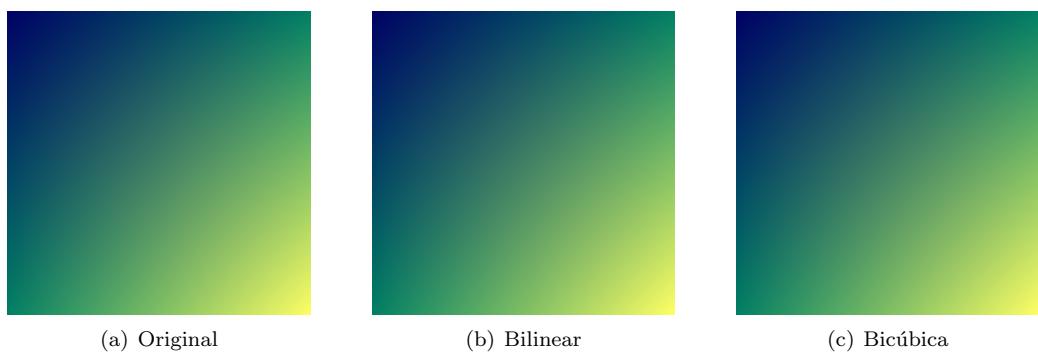


Figura 7: $f(x, y) = (x * y, x + y, 100)$

Na imagem 7, o erro foi $6.2793e^{-4}$ para a bilinear e $5.7830e^{-4}$ para a imagem bicubica.

4.1.3 Conclusões

As imagens em preto e branco geradas a partir de matrizes de 1 dimensão (ou seja, imagens cujas 3 matrizes RGB têm valores iguais) aparentam possuir comportamento melhor do que as imagens coloridas (que possuem valores diferentes para cada dimensão). Contudo, as matrizes em grayscale geradas a partir de imagens RGB têm erro maior do que imagens originalmente pigmentadas. Isso pode ser observado nas figuras 7 e 4.

No panorama geral, contudo, tanto as imagens coloridas quanto as imagens em preto e branco se comportam bem à interpolação. Comparativamente, o fator **cor** não afeta em peso a transformação da imagem como outras características estudadas neste EP.

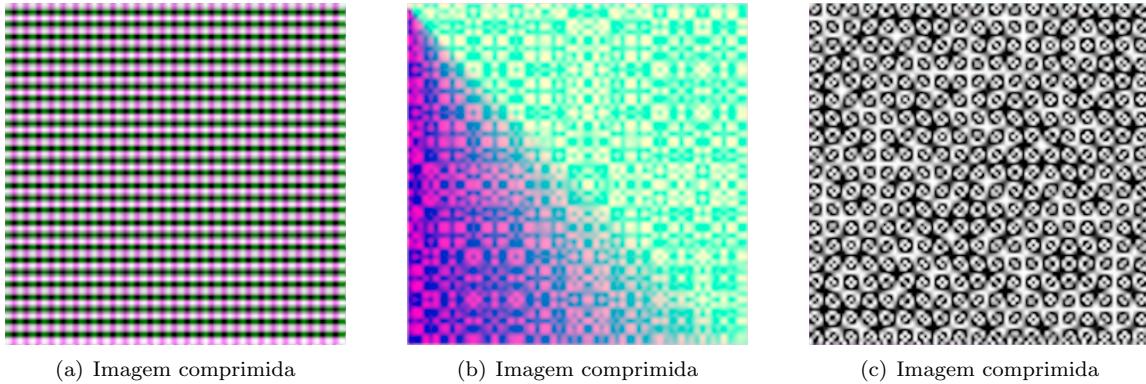


Figura 8: Imagens comprimidas, expandidas por meio do *GIMP*

Nessa linha de pensamento, era esperado que as imagens construídas a partir das funções (de classe C^2) obtivessem resultados mais satisfatórios do que os aqui encontrados. Como se sabe, o que os métodos aqui implementados fazem é "ampliar" a imagem comprimida. Assim, é possível observar que as imagens comprimidas têm comportamento de "zoom" nas originais, fazendo com que a descompressão atue na ampliação da imagem comprimida, como esperado. Portanto, os erros exorbitantes ocorrem não por causa dos métodos de interpolação implementados, mas sim pela perda de informações na compressão da imagem original. Isso é ainda mais visível para as funções periódicas, ou seja, a maioria das que foram implementadas para este EP.

Conclui-se, portanto, que o **tipo** de função construída é um fator relevante no resultado final da transformação das imagens.

Em relação ao parâmetro h , não observamos diferenças significativas ao alterá-lo.

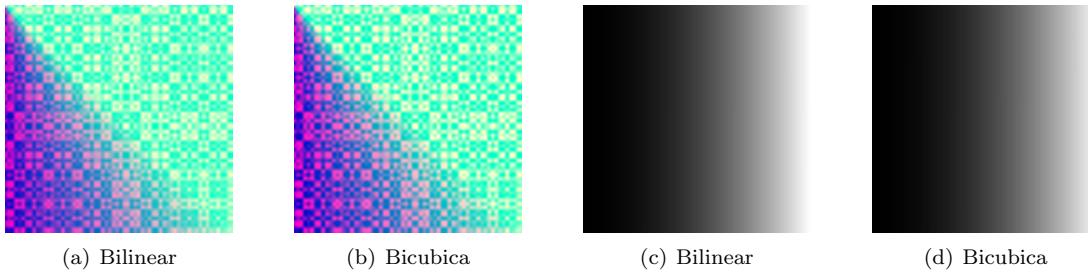


Figura 9: Imagens comprimidas e descomprimidas 3 vezes com $k = 1$

O parâmetro k , ao ser utilizado 3 vezes como $k = 1$, em comparação com a transformação por $k = 7$, tem resultados diferentes para imagens coloridas e imagens em preto e branco.

Como exemplo de imagem colorida, o erro encontrado das figuras 9(a) e 9(b) é de 0.17206 e de 0.17201, respectivamente. Portanto, os erros são menores para $k = 1$ interpolado 3 vezes quando comparado à imagem interpolada com $k = 7$. Já para imagens em preto e branco, as figuras 9(c) e 9(d) possuem erro de 0.0050739 e 0.0046943, respectivamente - logo, erro muito superior ao da imagem interpolada com $k = 7$.

Finalmente, no quesito método de interpolação, pode-se observar pelas imagens estudadas que o método bicubico possibilita mudanças menos bruscas na intensidade das cores e nos contornos das imagens descomprimidas. Isso é especialmente aparente na função $f(x, y) = \sin(xy)$. Isso não significa, porém, que os erros resultantes do método bicubico são menores que os do método bilinear. Em muitos dos exemplos expostos, o uso da interpolação bilinear produz melhores aproximações quando comparada à bicubica.

4.2 Parte 2: Selva

Testamos o programa com imagens reais, ou seja, cujas matrizes não foram geradas por funções:

4.2.1 Exemplos de imagens

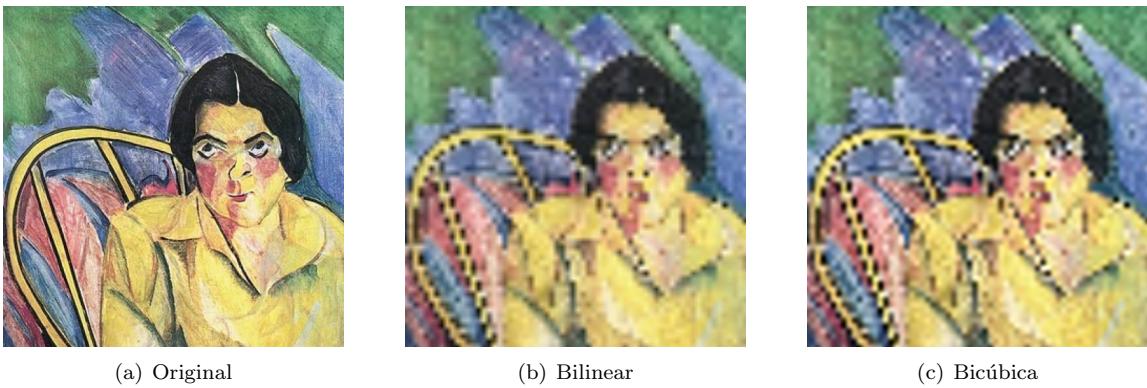


Figura 10: A Boba, de Anita Malfatti

Na figura 10, obtivemos um erro de 0.041672 para a bilinear e de 0.044602 para a imagem bicubica.

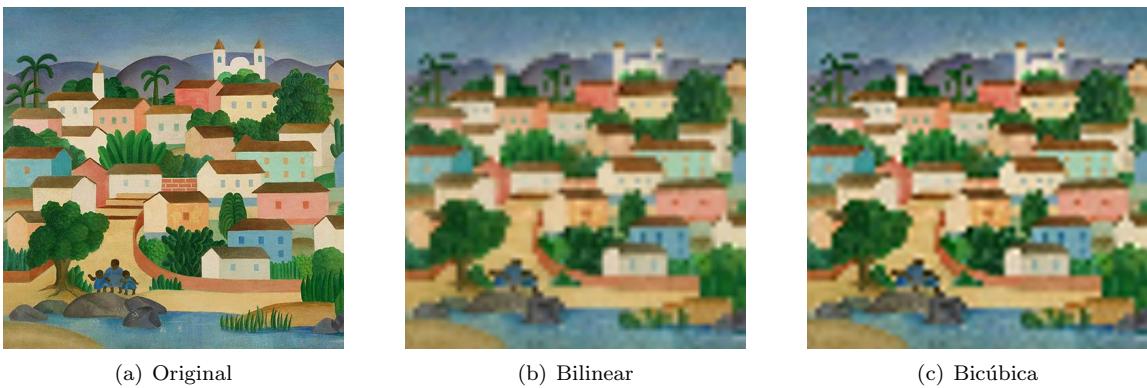


Figura 11: Obra de Tarsila do Amaral

Na figura 11, obtivemos um erro de 0.050520 para a bilinear e de 0.052251 para a imagem bicubica.



Figura 12: After Monet, de Pol Ledent

Na imagem 12, obtivemos um erro de 0.072162 para a bilinear e de 0.074628 para a imagem bicubica.

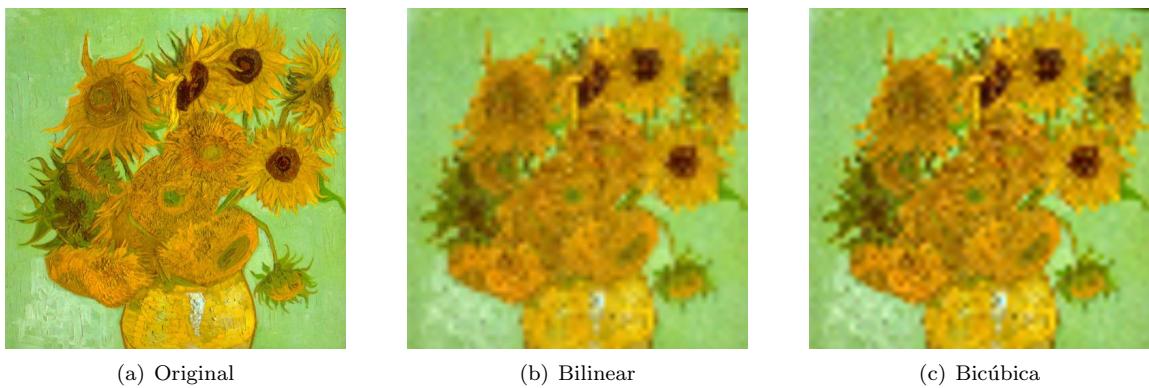


Figura 13: Doze Girassóis numa Jarra, de Van Gogh

Na figura 13, obtivemos um erro de 0.031110 para a bilinear e de 0.031818 para a imagem bicubica.

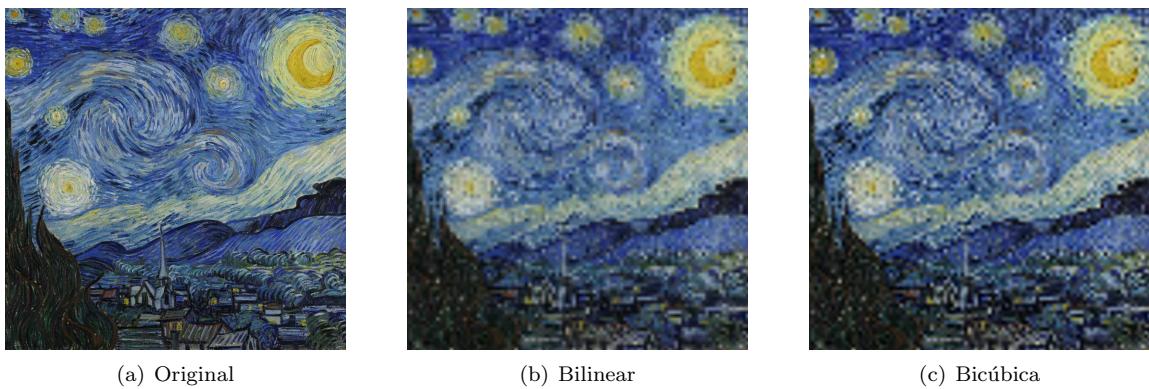


Figura 14: Noite Estrelada, de Van Gogh

Na imagem 14, obtivemos um erro de 0.036464 para a bilinear e de 0.038743 para a imagem bicubica.



Figura 15: A Boba em grayscale

Na figura 15, obtivemos um erro de 0.039875 para a bilinear e de 0.042767 para a imagem bicubica.

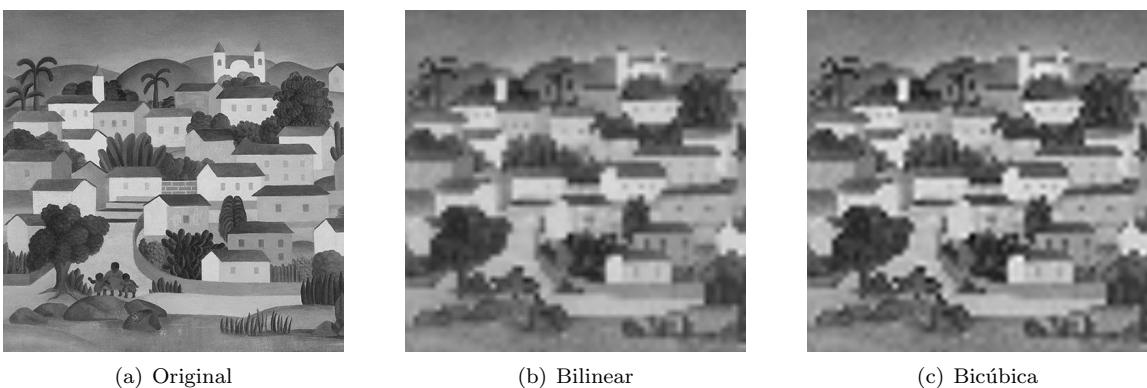


Figura 16: Obra de Tarsila do Amaral em grayscale

Na figura 16, obtivemos um erro de 0.048310 para a bilinear e de 0.050704 para a imagem bicubica.

4.2.2 Conclusões

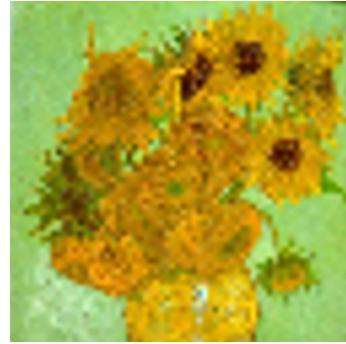
Como é possível observar, em geral, a interpolação funciona bem tanto para imagens coloridas, quanto para imagens em preto e branco. Além disso, a partir dos experimentos realizados, conclui-se que imagens reais possuem maior erro quando descomprimidas por interpolação bicubica se comparadas à descompressão por método bilinear. Visualmente, porém, os produtos finais das interpolações assemelham-se muito.

Diferentemente do caso das imagens geradas por funções C^2 , as imagens alteradas para preto e branco dos quadros de Anita Malfatti e Tarsila do Amaral expõem a preferência para interpolação de imagens originalmente coloridas na sua versão grayscale, visto que os erros encontrados para as mesmas são menores na versão não-RGB. Entretanto, visualmente, as versões coloridas são mais agradáveis.

Assim como para as imagens construídas por funções, a mudança do parâmetro h não resultou em diferenças expressivas na interpolação das imagens. Isso pode ser observado nas figuras 17, cujos erros foram 0.031818 para ambas as imagens. As duas imagens foram interpoladas pelo método bicúbico.



(a) $h = 0.01$



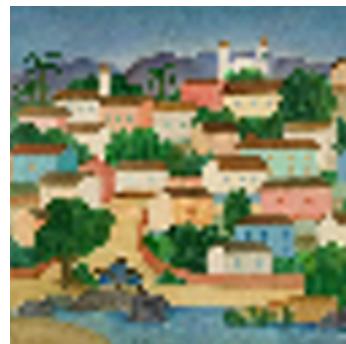
(b) $h = 1000$

Figura 17: Girassóis interpolados com h 's diferentes

Em relação aos testes com parâmetros k diferentes, comparando a imagem de Tarsila originalmente interpolada com $k = 7$ à mesma imagem interpolada 3 vezes com $k = 1$, o resultado obtido é que a interpolação feita mais vezes com um k menor apresenta melhor aproximação à imagem original, visto que o erro encontrado é menor - 0.052240 para a bilinear e 0.050512 para a bicubica.



(a) Bilinear



(b) Bicubica

Figura 18: Figuras interpoladas 3 vezes com $k = 1$