Introdução

Para este **EP** implementamos funções que comprimem e descomprimem imagens utilizando os métodos de interpolação bilinear e bicúbica. Com isso testamos os métodos em imagens geradas por funções e depois por imagens comuns.

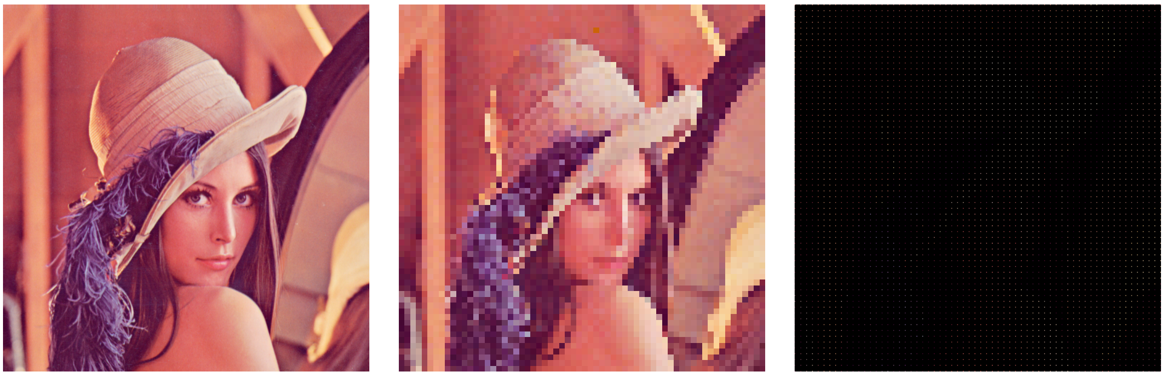
Implementação

Compressão, expansão e erro

Primeiramente implementamos as funções de compressão e expansão da imagem: dependentes do fator de compressão k:

```
function compress (originalImg, k)
function B = expand(A, k)
```

A função *compress* remove k linhas e/ou colunas entre cada pixel mantido, para isso é amostrado um pixel a cada $k+1$ linhas/colunas.
A função *expand* adiciona k linhas e colunas vazias entre cada pixel da imagem **A**, que serão preenchidas pela interpolação escolhida, e retorna a nova imagem em **B**.



(a) Imagem original (b) Imagem comprimida $k = 7$ (c) Imagem expandida $k = 7$

Figura 1: Primeiros processamentos

Implementamos também uma função que calcula o erro das imagens após o processo de compressão e descompressão comparando as diferenças pixel a pixel.

```
function err = calculateError(originalImg, decompressedImg)
```

Utilizando a seguinte relação:

$$err = \frac{errR + errG + errB}{3}, \text{ onde } errX = \frac{||origX - decX||_2}{||origX||_2}, \; X \in \{R, G, B\}$$

Bilinear

Para a interpolação bilinear implementamos o polinômio aproximador:

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

Usando os valores dos cantos dos quadrados da imagem original como $\{x_i, y_j; x_{i+1}, y_{j+1}\}$. Desta forma calculamos os coeficientes resolvendo o seguinte sistema linear:

$$\begin{bmatrix} f(x_i, y_j) \\ f(x_i, y_{j+1}) \\ f(x_{i+1}, y_j) \\ f(x_{i+1}, y_{j+1}) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & h & 0 \\ 1 & h & 0 & 0 \\ 1 & h & h & h^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Bicúbico

Para a interpolação bicúbica utilizamos o seguinte polinômio:

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

Usando os valores dos cantos dos quadrados da imagem original como $\{x_i, y_j; x_{i+1}, y_{j+1}\}$. Para obter os valores a_{ij} resolvemos o seguinte sistema:

$$\begin{bmatrix} f(x_i, y_j) & f(x_i, y_{j+1}) & \frac{\partial f}{\partial y}(x_i, y_j) & \frac{\partial f}{\partial y}(x_i, y_{j+1}) \\ f(x_{i+1}, y_j) & f(x_{i+1}, y_{j+1}) & \frac{\partial f}{\partial y}(x_{i+1}, y_j) & \frac{\partial f}{\partial y}(x_{i+1}, y_{j+1}) \\ \frac{\partial f}{\partial x}(x_i, y_j) & \frac{\partial f}{\partial x}(x_i, y_{j+1}) & \frac{\partial f}{\partial x \partial y}(x_i, y_j) & \frac{\partial f}{\partial x \partial y}(x_i, y_{j+1}) \\ \frac{\partial f}{\partial x}(x_{i+1}, y_j) & \frac{\partial f}{\partial x}(x_{i+1}, y_{j+1}) & \frac{\partial f}{\partial x \partial y}(x_{i+1}, y_j) & \frac{\partial f}{\partial x \partial y}(x_{i+1}, y_{j+1}) \end{bmatrix} = B \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} B^T$$

onde,

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

Zoológico

Para esta parte do EP utilizamos as seguintes funções para gerar as imagens de teste:

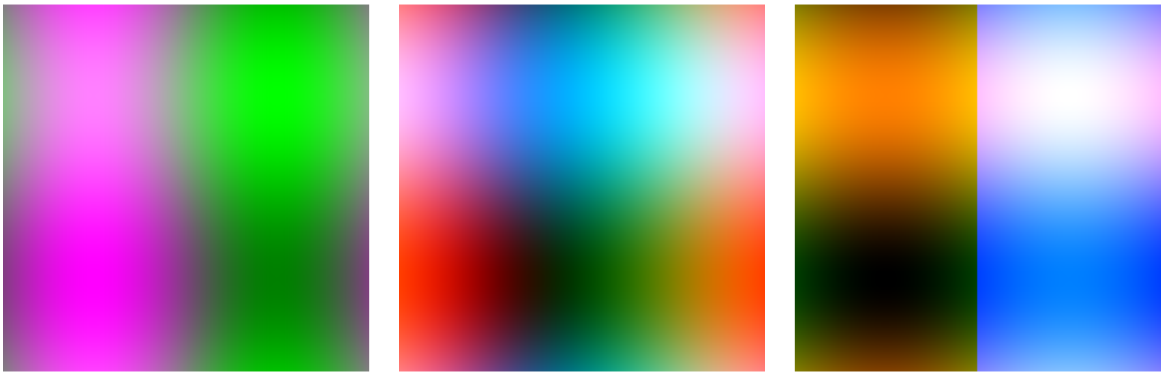
$$f_1(x, y) = \left(\sin(x), \frac{\sin(y) - \sin(x)}{2}, \sin(x) \right)$$

$$f_2(x, y) = \left(\cos(x), \frac{\sin(y) - \sin(x)}{2}, \sin(y) \right)$$

$$f_3(x, y) = \left(\sin(y), \frac{\sin(y) - \sin(x)}{2}, \begin{cases} 0, & x \leq \pi \\ 1, & x > \pi \end{cases} \right)$$

Geramos imagens de 281×281 pixels. Para isso foi feita a transformação de pixels de 0 a 2π e a saída das funções para 0 a 1 para geração das imagens. Note que $f_3 \notin C^2$.

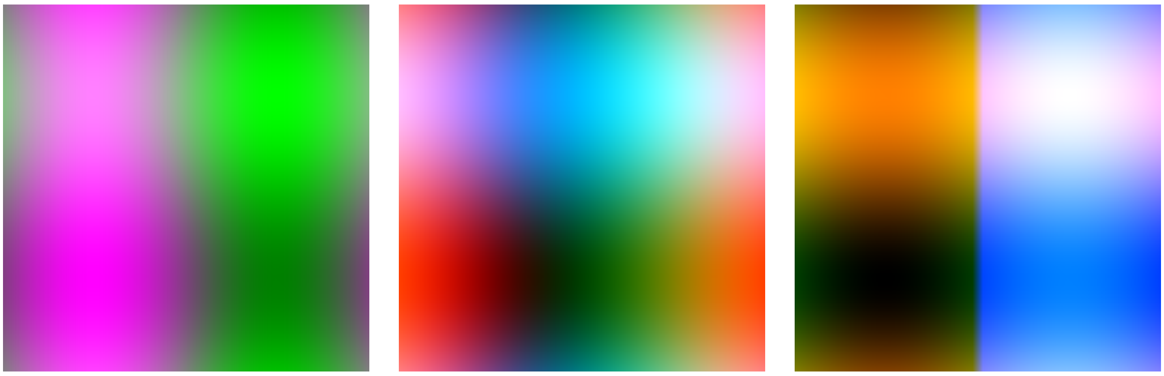
As funções geraram as seguintes imagens:



(a) Imagem de $f_1(x, y)$ (b) Imagem de $f_2(x, y)$ (c) Imagem de $f_3(x, y)$

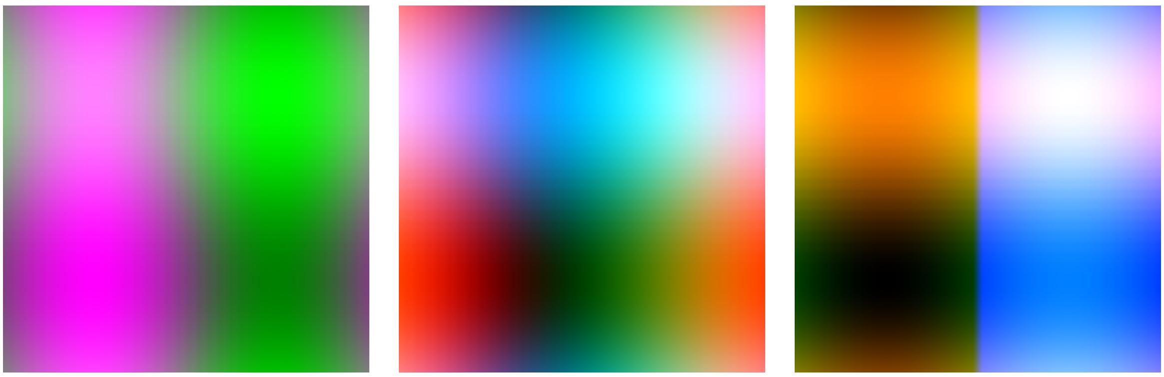
Figura 2: Imagens originais Zoológico

As saídas após a compressão e descompressão das imagens com $k = 7$ junto a seus respectivos erros são:



(a) $err = 0.0014335$ (b) $err = 0.0014313$ (c) $err = 0.024147$

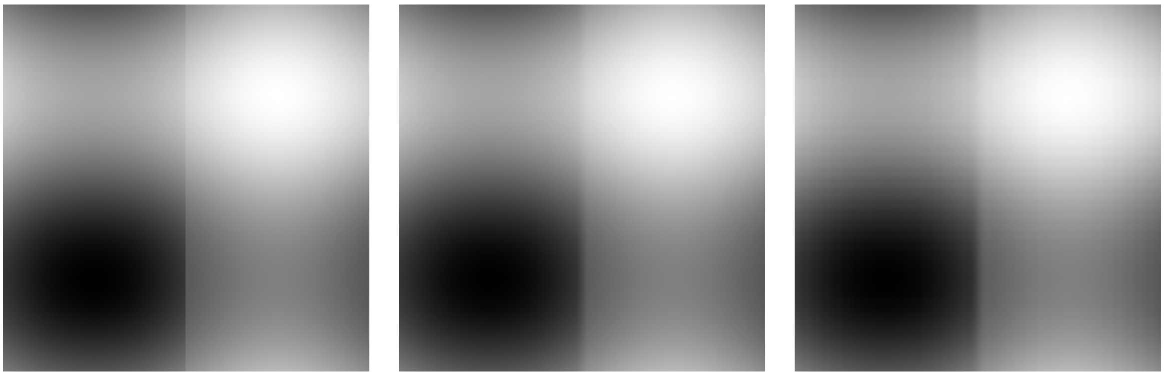
Figura 3: Imagens descomprimidas utilizando o método bilinear



(a) $err = 0.0043710$ (b) $err = 0.0043806$ (c) $err = 0.024140$

Figura 4: Imagens descomprimidas utilizando o método bicúbico

Também analisamos a descompressão utilizando um versão preto e branco da imagem de $f_3(x, y)$ cujos resultados foram:



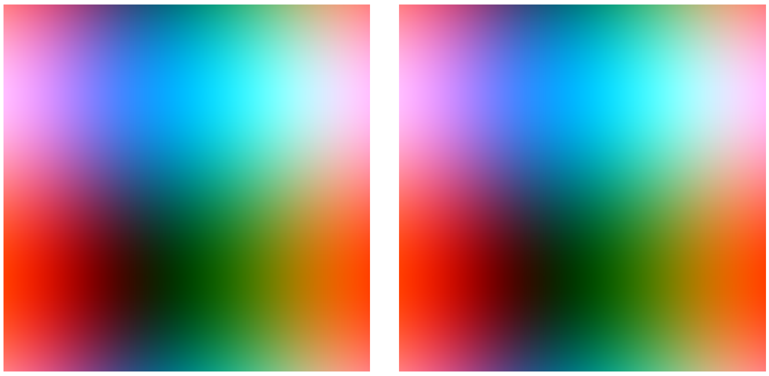
(a) Imagem original em PB (b) Bilinear: $err = 0.0070867$ (c) Bicúbico: $err = 0.0070402$

Figura 5: Imagens preto e branco

Após análise da imagens podemos responder que:

- Funciona bem para imagens preto e branco?
  - As imagens geradas ficaram bem próximas da original usando tanto o método bilinear quanto o bicúbico, com $err < 0.01$ para ambos.
- Funciona bem para imagens coloridas?
  - As imagens geradas ficaram bem próximas da original usando tanto o método bilinear quanto o bicúbico, com $err < 0.01$ para as funções $\in C^2$ e $err < 0.1$ para f_3.
- Funciona bem para todas as funções de classe C^2?
  - Tivemos erros muito pequenos para todas as funções de C^2 e as imagens ficaram visualmente muito parecidas.
- E para funções que não são de classe C^2?
  - Tivemos erros muito pequenos para todas as funções que não são C^2, porém é possível notar que a mudança drástica de cor gerada pela discontinuidade de f_3 foi "perdida"pois as interpolações suavizam a mudança.
- Como se comporta o erro?
  - Tivemos erros muito pequenos para todas imagens, sempre com $err < 0.1$.

Podemos também comparar a decompressão direta usando $k = 7$ ou três descompressões seguidas com $k = 1$ obtendo assim:



(a) $errOrig = 0.0014338$
$errK7 = 0.000013017$

(b) $errOrig = 0.0010179$
$errK7 = 0.0042732$

Figura 6: Imagens com múltiplas descompressões com $k = 1$
(a): Bilinear (b): Bicúbico

Podemos observar que no caso bilinear a imagem ficou muito próxima da versão com $k = 7$, obtendo um erro com o original bem próximo do primeiro caso e com um erro da ordem de 10^{-4} quando comparado com a primeira descompressão.
Para o caso bicúbico podemos notar que a diferença entre imagens descomprimidas foi mais significativa, o que refletiu num erro menor quando comparado com a imagem original.

Selva

Para as análises da selva utilizamos a famosa imagem conhecida por *lena* comunmente usada em testes e demonstrações de análise de imagem. A imagem original tem dimensões 512×512 pixels porém por razões de redimensionamento cortamos a imagem para que tivesse dimensões 505×505 pixels.

Assim geramos as seguintes imagens e análises:

(a) Imagem original (b) Bilinear $err = 0.032717$ (c) Bicúbico $err = 0.033258$

Figura 7: Imagens *lena*, usando $k = 7$

(a) Imagem original

(b) Bilinear $err = 0.031117$

(c) Bicúbico $err = 0.031519$

Figura 8: Imagens *lena* em PB, usando $k = 7$

- Funciona bem para imagens preto e branco?
  - As imagens geradas ficaram visualmente diferentes da original podendo ser notada uma perda de qualidade da imagem, porém as imagens em preto e branco não apresentaram diferença na qualidade quando comparadas com suas versões coloridas.
- Funciona bem para imagens coloridas?
  - Como mencionado acima tivemos perda de qualidade das imagens descomprimidas que foram bem mais notáveis do que nos casos do Zoológico, com $err \approx 0.03$.
- Como se comporta o erro?
  - Tivemos erros pequenos para todas imagens, sempre com $err < 0.1$.

(a) $errOrig = 0.032759$
$errK7 = 0.0036101$

(b) $errOrig = 0.033180$
$errK7 = 0.0073432$

Figura 9: Imagens com múltiplas descompressões com $k = 1$
(a): Bilinear (b): Bicúbico

Podemos notar que ambas as imagens geradas com múltiplas descompressões com $k = 1$ ficaram bem próximas das geradas com $k = 7$, com $errK7 < 0.1$, e com o $errOrig$ muito próximos dos erros da descompressão simples.