



Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Engenharia



Programa de Graduação em Engenharia da Computação

Zoom In: Interpolação Gráfica e Aliasing

Aluno: Maiki Buffet

Professora: Beatriz R. Tavares Franciosi

Porto Alegre

Junho, 2017

Sumário

A. Introdução.....	3
B. Conceito.....	3
C. Tipos de Algoritmos.....	4
D. Aplicação.....	7
E. Consequências da Interpolação.....	7
F. Anti-Aliasing.....	7
G. Escolha do Algoritmo.....	8
H. Implementação do Algoritmo.....	9
I. Resultado.....	12
J. Compilação / Execução.....	13
K. Referências Bibliográficas.....	13

A. Introdução

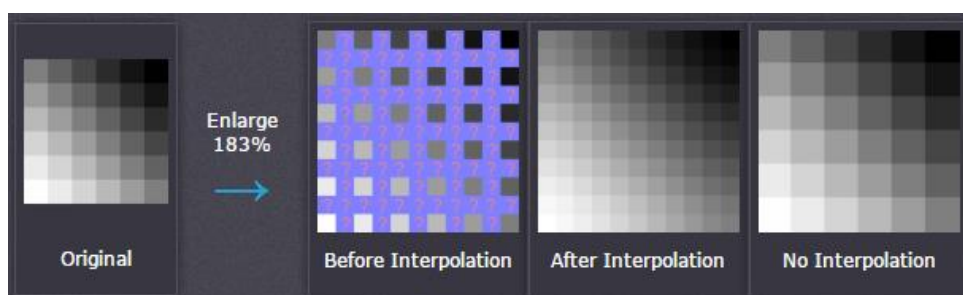
A interpolação de imagem ocorre em todas as fotos digitais em algum estágio - seja em processos de reconstrução ou ampliação de foto. Isso acontece sempre que uma imagem for redimensionada ou remapeada (distorcida) de uma grade de pixels para outra. O redimensionamento da imagem é necessário quando se precisa aumentar ou diminuir o número total de pixels, enquanto o remapeamento pode ocorrer em uma variedade maior de cenários: corrigir a distorção da lente, mudar a perspectiva ou girar uma imagem.



Mesmo que o mesmo redimensionamento ou remapeamento da imagem seja realizado, os resultados podem variar significativamente de acordo com o algoritmo de interpolação. É apenas uma aproximação, portanto, uma imagem sempre perderá alguma qualidade independente da interpolação realizada.

B. Conceito

A interpolação de imagem funciona em duas direções e tenta obter uma melhor aproximação da cor e intensidade de um pixel com base nos valores dos pixels circundantes. O exemplo a seguir ilustra como o redimensionamento/ampliação funciona:



Ao contrário do gradiente ideal acima, os valores de pixels podem variar muito mais rapidamente de um local para o outro. Quanto mais se conhece os pixels circundantes, melhor será a interpolação. Portanto, os resultados se deterioram rapidamente quanto mais se estica uma imagem e a interpolação nunca pode adicionar detalhes à uma imagem que ainda não está presente.

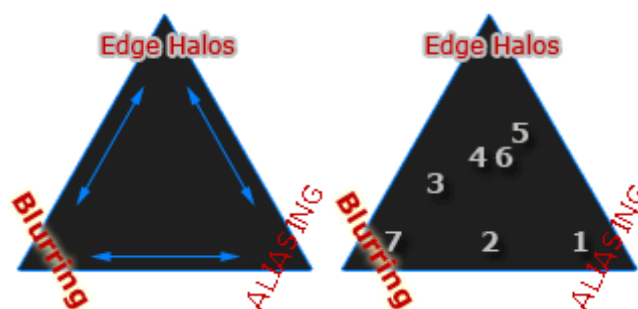
C. Tipos de Algoritmos

Os algoritmos de interpolação comuns podem ser agrupados em duas categorias: adaptativos e não-adaptativos. Os métodos adaptativos mudam dependendo do que estão interpolando (bordas afiadas versus textura suave), enquanto que os métodos não adaptativos tratam todos os pixels de forma igual.

Algoritmos adaptativos: incluem muitos algoritmos proprietários em software licenciado, tais como: Qimage, PhotoZoom Pro, Fractals Genuine e outros. Muitos destes aplicam uma versão diferente de seu algoritmo (em uma base de pixel por pixel), quando detectam a presença de uma borda - visando minimizar os artefatos de interpolação desagradáveis em regiões onde são mais aparentes. Esses algoritmos são projetados principalmente para maximizar o detalhe livre de artefatos nas fotos ampliadas, de modo que alguns não podem ser usados para distorcer ou girar uma imagem.

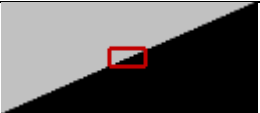
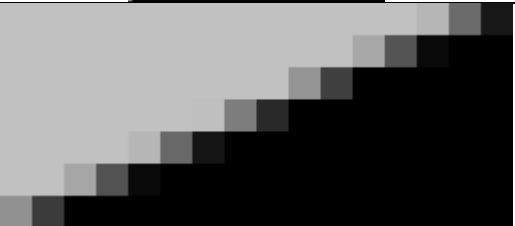




Algoritmos não-adaptativos: nearest neighbor, bilinear, bicubic, spline, sinc, lanczos e outros. Dependendo da sua complexidade, estes usam de 0 a 256 (ou mais) pixels adjacentes ao interpolar. Os pixels mais adjacentes que eles incluem, mais precisos podem se tornar, mas isso ocorre à custa de um tempo de processamento muito maior. Esses algoritmos podem ser usados para distorcer e redimensionar uma foto.



- Todos os algoritmos de interpolação não-adaptativos sempre enfrentam um *trade-off* entre três artefatos: aliasing, blurring e edge halos. O diagrama a seguir e a comparação visual demonstram onde cada algoritmo está neste truque de guerra de três vias.



O diagrama qualitativo para a direita demonstra de forma aproximada os *trade-offs* de cada tipo (apresentados logo em

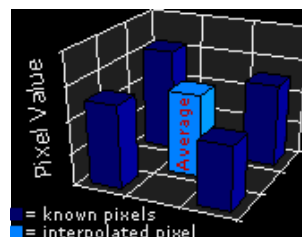
seguida). O nearest neighbor é o que possui mais aliasing, e junto com o bilinear, estes são os únicos dois que não possuem edge halos - apenas um equilíbrio diferente de aliasing e blurring. Pode-se observar que a nitidez da borda aumenta gradualmente de 3-5, mas à custa dos aumentos maiores de aliasing e edge halos. Lanczos é muito semelhante ao bicubic e bicubic smoother, exceto talvez um pouco mais aliased. Todos mostram algum grau de aliasing, no entanto, sempre podemos eliminar aliasing inteiramente ao utilizarmos desfocagem da imagem em software.

Imagem Original	
1-Nearest Neighbor	
2-Bilinear	
3-Bicubic Smoother	
4-Bicubic	
5-Bicubic Sharper	

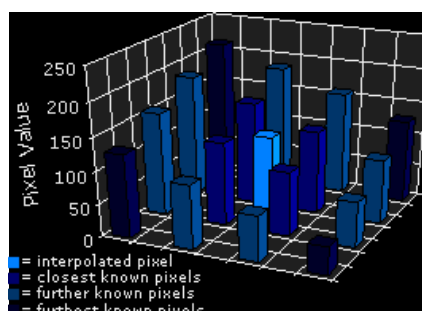
6-Lanczos	
7-Bilinear w/ Blur	

Nearest Neighbor: é o mais básico e requer o menor tempo de processamento de todos os algoritmos de interpolação porque ele só considera um pixel - o mais próximo ao ponto interpolado. Isso tem o efeito de simplesmente tornar cada pixel maior.

Bilinear: considera o quadrante 2x2 mais próximo de valores de pixel conhecidos em torno do pixel desconhecido. Em seguida, leva uma média ponderada desses 4 pixels para chegar ao seu valor interpolado final. Isso resulta em imagens muito mais lisas do que o nearest neighbor.



Bicubic: vai um passo além do bilinear considerando o quadrante 4x4 mais próximo de pixels conhecidos - para um total de 16 pixels. Uma vez que estas estão a várias distâncias do pixel desconhecido, os pixels mais próximos recebem uma maior ponderação no cálculo. Este, produz imagens visivelmente mais nítidas que os dois métodos anteriores e talvez seja a combinação ideal de tempo de processamento e qualidade de saída. Por esse motivo, é um padrão em muitos programas de edição de imagem, drivers de impressora e interpolação na câmera.



De ordem máxima (Spline): são extremamente úteis quando a imagem requer rotações/distorções múltiplas em etapas separadas. No entanto, para ampliações ou rotações em um único passo, esses algoritmos de ordem superior proporcionam uma melhoria visual decrescente à medida que aumenta o tempo de processamento.

D. Aplicação

O dimensionamento de imagem é usado, entre outras aplicações, em navegadores web, editores de imagens, visualizadores de imagens e arquivos, lupas de software, zoom digital, o processo de geração de imagens em miniatura e a saída de imagens através de telas ou impressoras.

E. Consequências da Interpolação

Todos os interpoladores não-adaptativos tentam encontrar um equilíbrio ótimo entre três artefatos indesejáveis: edge halos, blurring e aliasing.



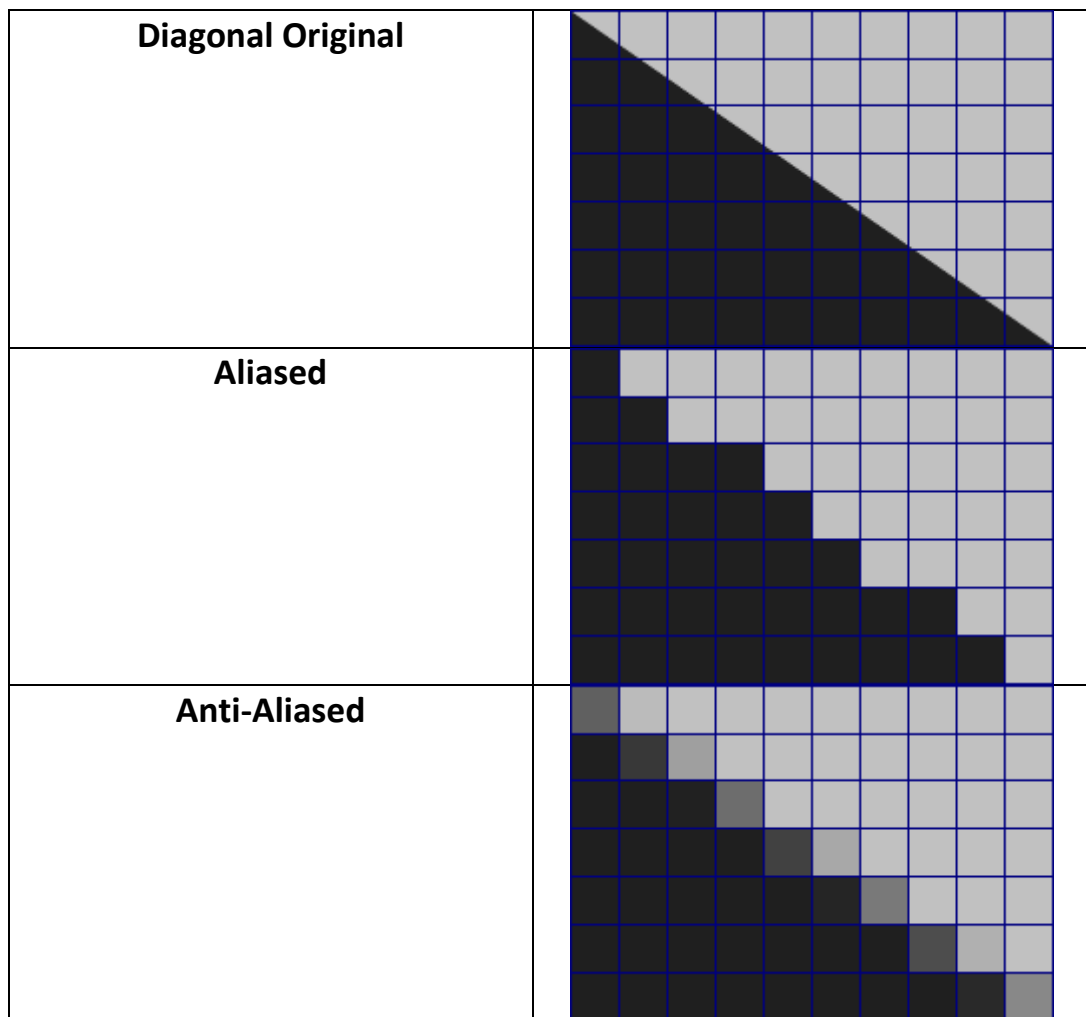
Mesmo os interpoladores não-adaptativos mais avançadas sempre têm que aumentar ou diminuir um dos artefatos acima à custa dos outros dois - portanto, pelo menos, um será visível.

F. Anti-Aliasing

Anti-aliasing é um processo que tenta minimizar a aparência de bordas diagonais irregulares, denominadas "jaggies". Isso dá ao texto ou às imagens uma aparência digital áspera:



Anti-aliasing remove esses "jaggies" e dá a aparência de bordas mais suaves e maior resolução. Ele funciona levando em consideração o quanto uma borda ideal se sobrepõe a pixels adjacentes. A borda aliased simplesmente arredonda para cima ou para baixo sem valor intermediário, enquanto a borda anti-alias fornece um valor proporcional ao quanto da borda estava dentro de cada pixel:



G. Escolha do Algoritmo

O algoritmo bicúbico (tópico E, exemplo 4 da imagem abaixo), foi escolhido para a implementação do *zoom in* pois é um dos mais utilizados na área de redimensionamento de imagem, por ser muito

equilibrado no processamento dos três artefatos apresentados anteriormente (halo edge, blurring e aliasing) e possuir um “custo versus benefício” (em termos de processamento e qualidade de imagem) muito satisfatório, se comparado aos outros métodos apresentados anteriormente.



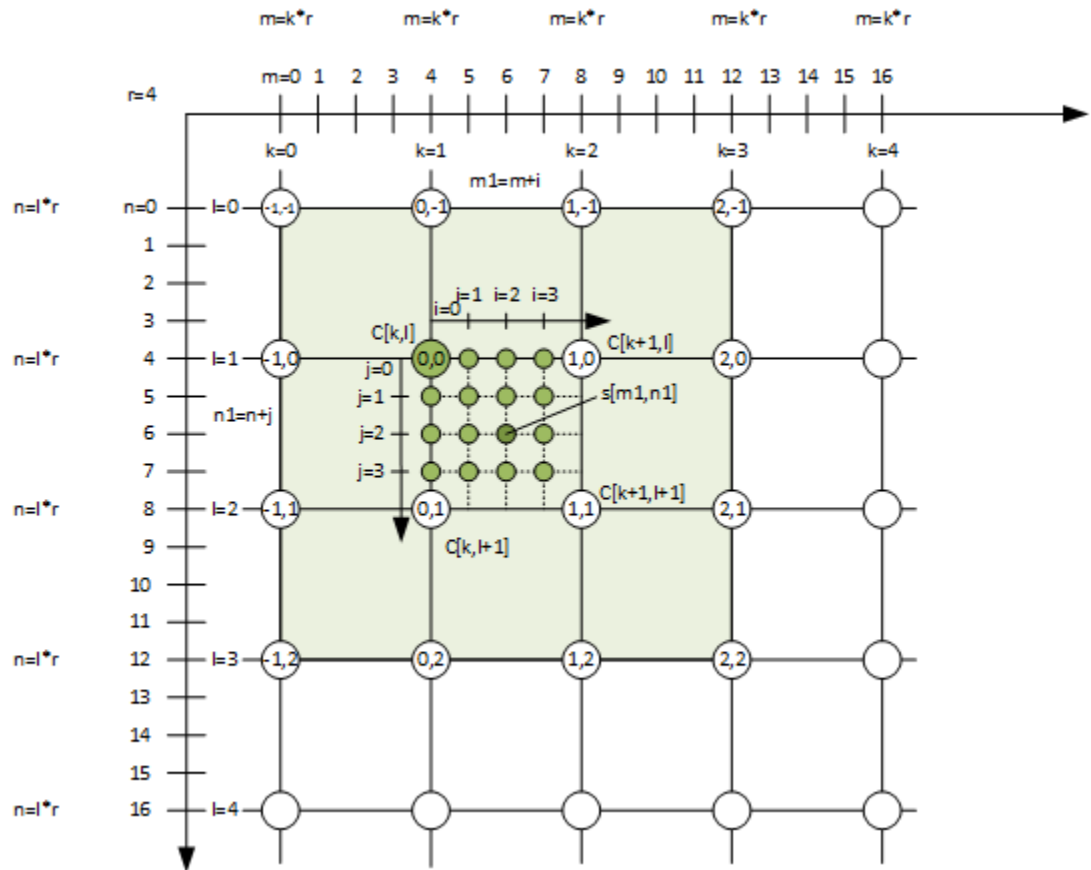
H. Implementação do Algoritmo

A interpolação bicúbica é nada mais que uma interpolação cúbica em duas dimensões – horizontal e vertical. É importante ressaltar que a quantidade do número de dimensões não necessariamente afeta o algoritmo, mas sim a quantidade de vezes que deverá ser aplicado.

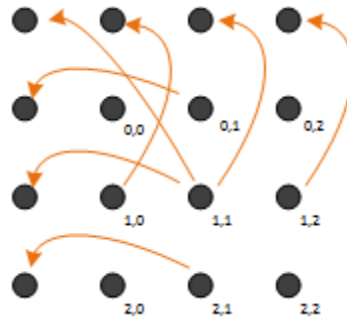
A fórmula geral utilizada pela grande maioria dos filtros bicúbicos é:

$$k(x) = \frac{1}{6} \begin{cases} (12 - 9B - 6C)|x|^3 + (-18 + 12B + 6C)|x|^2 + (6 - 2B) & \text{if } |x| < 1 \\ (-B - 6C)|x|^3 + (6B + 30C)|x|^2 + (-12B - 48C)|x| + (8B + 24C) & \text{if } 1 \leq |x| < 2 \\ 0 & \text{otherwise} \end{cases}$$

Supondo que tenhamos 16 pontos $F(p,q)$ de uma matriz qualquer (representada logo abaixo), com p e q sendo intercalados de 0 a 3 com $F(p,q)$ localizado em $(p-1, q-1)$. Interpolaremos a área $[0,1] \times [0,1]$, iniciando a interpolação, primeiramente, pelas quatro colunas e o resultado é interpolado na horizontal (quatro linhas).



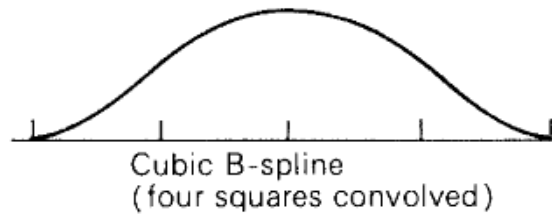
O algoritmo desenvolvido captura as coordenadas da imagem (X,Y) e retorna o valor interpolado dentro dos 16 pontos mais próximos a ele. Os 16 pontos são iterados por 2 loops de $[x-1,y-1]$ até $[x+2,y+2]$.



Para cada elemento próximo, um fator de interpolação é calculado com a correspondente função de interpolação (neste caso bicúbica BSpline).

Para a criação da função Spline, fora utilizada a seguinte equação:

$$s(m_1, n_1) = \sum_{a=-1}^2 \sum_{b=-1}^2 c(k+a, l+b) \beta^3 \left(\frac{m_1}{r} - k - a \right) \beta^3 \left(\frac{n_1}{r} - l - b \right),$$



O gráfico acima da função Spline(x), é plotado na direção de Y. O x inicia em -2 e termina em +2. Isto indica que quanto maior for o valor de x, a função é mais lenta e, portanto, dados mais afastados recebem menor peso. A imagem de destino é uma cópia da imagem original, “traduzida” e redimensionada. As coordenadas x,y na imagem fonte são calculadas para cada destino dos novos pixels (pixels que serão interpolados). Encontra-se o pixel central (dentro todos os 16), e a interpolação é executada. Fazendo-se os cálculos para aproximação do algoritmo, resulta-se na função abaixo:

```
float CubicHermiteSpline(float A, float B, float C, float D, float t) {
    float a = -A / 2.0f + (3.0f * B) / 2.0f - (3.0f * C) / 2.0f + D / 2.0f;
    float b = A - (5.0f * B) / 2.0f + (2.0f * C) - D / 2.0f;
    float c = -A / 2.0f + C / 2.0f;
    float d = B;
    return a*t*t*t + b*t*t + c*t + d;
}

const uint8_t* GetPixelClamped(const ImageData& image, int x, int y) {

array<uint8_t, 3> SampleBicubic(const ImageData& image, float u, float v) {
    float x = (u * image.m_width) - 0.5;
    int xint = int(x);
    float xfraction = x - floor(x);
    float y = (v * image.m_height) - 0.5;
    int yint = int(y);
    float yfraction = y - floor(y);
    array<uint8_t, 3> ret;

    auto p00 = GetPixelClamped(image, xint - 1, yint - 1);
    auto p10 = GetPixelClamped(image, xint + 0, yint - 1);
    auto p20 = GetPixelClamped(image, xint + 1, yint - 1);
    auto p30 = GetPixelClamped(image, xint + 2, yint - 1);
    auto p01 = GetPixelClamped(image, xint - 1, yint + 0);
    auto p11 = GetPixelClamped(image, xint + 0, yint + 0);
    auto p21 = GetPixelClamped(image, xint + 1, yint + 0);
    auto p31 = GetPixelClamped(image, xint + 2, yint + 0);
    auto p02 = GetPixelClamped(image, xint - 1, yint + 1);
    auto p12 = GetPixelClamped(image, xint + 0, yint + 1);
    auto p22 = GetPixelClamped(image, xint + 1, yint + 1);
    auto p32 = GetPixelClamped(image, xint + 2, yint + 1);
    auto p03 = GetPixelClamped(image, xint - 1, yint + 2);
    auto p13 = GetPixelClamped(image, xint + 0, yint + 2);
    auto p23 = GetPixelClamped(image, xint + 1, yint + 2);
    auto p33 = GetPixelClamped(image, xint + 2, yint + 2);

    for(int i = 0; i < 3; ++i) {
        float col0 = CubicHermiteSpline(p00[i], p10[i], p20[i], p30[i], xfraction);
        float col1 = CubicHermiteSpline(p01[i], p11[i], p21[i], p31[i], xfraction);
        float col2 = CubicHermiteSpline(p02[i], p12[i], p22[i], p32[i], xfraction);
        float col3 = CubicHermiteSpline(p03[i], p13[i], p23[i], p33[i], xfraction);
        float value = CubicHermiteSpline(col0, col1, col2, col3, yfraction);
        if(value < 0.0f) value = 0.0f;
        else if(value > 255.0f) value = 255.0f;
        ret[i] = uint8_t(value);
    }
    return ret;
}
```

I. Resultado



Original: **100%**
Zoom In: **300%**



Como pode ser observado nos 4 testes realizados – redimensionando a foto em 300%, o método de interpolação bicúbico Spline não apresenta efeitos de aliasing, embora a imagem fique levemente “borrada”, a qualidade de *zoom in* é perceptível. O custo computacional deste método é alto, porém vide a qualidade de saída produzida, se comparado a outros métodos (previamente apresentados), pode-se dizer que é um dos melhores métodos de redimensionamentos a ser utilizado.

J. Compilação / Execução

O presente programa funciona apenas em sistemas operacionais Windows, pois utiliza da biblioteca *windows.h* para utilização dos cabeçalhos de bitmap.

Para a compilação é necessário passar os seguintes parâmetros:

- -lm
- -std=c++11

Exemplo: `g++ main.c -o main -lm -std=c++11`

Para a execução, são necessários quatro argumentos, sendo eles:

- Nome da imagem original (apenas no formato bmp 24bits)
- Nome da imagem redimensionada
- Escala de redimensionamento (ex: 3 = 300%)

Exemplo: `./main input.bmp output.bmp 3`

K. Referências Bibliográficas

www2.units.it/ipl/students_area/imm2/files/Numerical_Recipes.pdf

stackoverflow.com/questions/34047874/scipy-ndimage-interpolation-zoom-uses-nearest-neighbor-like-algorithm-for-scalin

stackoverflow.com/questions/12729228/simple-efficient-bilinear-interpolation-of-images-in-numpy-and-python

stackoverflow.com/questions/34622717/bicubic-interpolation-in-c

stackoverflow.com/questions/15176972/bi-cubic-interpolation-algorithm-for-image-scaling

github.com/yglukhov/bicubic-interpolation-image-processing/blob/master/bicubic03042002.pdf

github.com/mortennobel/java-image-scaling/blob/master/src/main/java/com/mortennobel/imagescaling/

docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.ndimage.interpolation.zoom.html

paulbourke.net/miscellaneous/interpolation/
imagemagick.org/Usage/filter/
imagemagick.org/Usage/resize/
imagemagick.org/Usage/filter/nicolas/
tech-algorithm.com/articles/nearest-neighbor-image-scaling/
tech-algorithm.com/articles/linear-interpolation/
tech-algorithm.com/articles/bilinear-image-scaling/
tech-algorithm.com/articles/trilinear-interpolation-image-scaling/
people.sc.fsu.edu/~jburkardt/c_src/nearest_interp_1d/nearest_interp_1d.html
codeproject.com/Articles/236394/Bi-Cubic-and-Bi-Linear-Interpolation-with-GLSL
math.ewha.ac.kr/~jylee/SciComp/dip-crane/
shulgadim.blogspot.com.br/2014/02/2d-cubic-b-spline-interpolation-via.html
scipy-lectures.org/advanced/image_processing/
docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html
pyimagesearch.com/2014/01/20/basic-image-manipulations-in-python-and-opencv-resizing-scaling-rotating-and-cropping/
docs.opencv.org/3.1.0/da/d6e/tutorial_py_geometric_transformations.html
entropymine.com/imageworsener/bicubic/
catenary.com/howto/enlarge.html
user.xmission.com/~legalize/zoom.html
user.xmission.com/~legalize/zoom.1.html
cambridgeincolour.com/tutorials/image-interpolation.htm
cambridgeincolour.com/tutorials/digital-photo-enlargement.htm
cambridgeincolour.com/tutorials/image-resize-for-web.htm
cplusplus.com/forum/general/2615/
cplusplus.com/forum/general/174907/
ida.upmc.fr/~zaleski/markers_doc/interpolation_8c-source.html
ida.upmc.fr/~zaleski/markers_doc/files.html
pastebin.com/sQDQg7SG
blog.demofox.org/2015/08/15/resizing-images-with-bicubic-interpolation/
homepages.inf.ed.ac.uk/rbf/BOOKS/PHILLIPS/cips2ed.pdf
duanqu.tech/questions/4330564/python-bilinear-image-interpolation
mrl.nyu.edu/~perlin/cubic/Cubic_java.html

convertio.co/pt/conversor-ppm/

people.cs.clemson.edu/~levinej/courses/S15/1020/handouts/lec05/creatingPPM.pdf

gamedev.net/topic/336744-bicubic-interpolation-on-image-enlargements/

caca.zoy.org/browser/libpipi/trunk/pipi/resample/bicubic.c?rev=4696

gamedev.net/topic/572816-bicubic-interpolation/

paulinternet.nl/?page=bicubic

en.wikipedia.org/wiki/Image_scaling

en.wikipedia.org/wiki/Bicubic_interpolation

en.wikipedia.org/wiki/Bilinear_interpolation

en.wikipedia.org/wiki/Nearest-neighbor_interpolation

en.wikipedia.org/wiki/Cubic_Hermite_spline

en.wikipedia.org/wiki/Hermite_interpolation

tanbakuchi.com/posts/comparison-of-openv-interpolation-algorithms/