PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

INTERPOLAÇÃO DE IMAGENS

FELIPE PFEIFER RUBIN

Orientador: Prof. Beatriz Franciosi

Porto Alegre
2017

INTERPOLAÇÃO DE IMAGENS

RESUMO

Ferramentas que utilizamos diariamente realizam diferentes operações matemáticas com o propósito de realizar o zoom digital em nossas imagens. Propõe-se utilizar do método de interpolação de Newton para realizar as operações de zoom-in (aumento) e zoom-out (diminuição) em uma dada imagem.

Palavras-Chave: interpolação, processamento de imagens, métodos numéricos, newton.

IMAGE INTERPOLATION

ABSTRACT

Daily we use tools that do different mathematical operation with the purpose of doing digital zoom in our images. It is proposed to utilize Newtons interpolating polynomial to apply zoom-in and zoom-out operations in a given image.

Keywords: Interpolation, image processing, numerical methods, Newton.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SIGLAS

PNG – Portable Network Graphics

LISTA DE ABREVIATURAS

LISTA DE SÍMBOLOS

SUMÁRIO

1. INTRODUÇÃO

O entendimento de diversos algoritmos presentes em ciência da computação têm como pré requisito o estudo teórico deste originado de outras áreas, como por exemplo a matemática. O zoom digital, uma operação que realiza o aumento e diminuição de uma imagem é feito através de métodos de interpolação que possuem base na matemática.

O objeto deste trabalho é complementar o estudo dos métodos de interpolação vistos em sala de aula demonstrando sua aplicação em operações que realizamos diariamente, como o zoom digital. Para a realização de tal operação deve-se utilizar do método de interpolação de Newton ou Spline sobre uma dada imagem colorida em formato png e retorná-la em grayscale (tons de cinza) após aplicado a operação de zoom. Além disso, deve-se decidir entre a análise de uma biblioteca que implementa tais operações ou de fato implementá-las. A decisão sobre tais opções de realização do trabalho foi de implementar do método de interpolação de Newton na linguagem de programação C++.

Nas seções seguintes, serão descritos a base matemática da interpolação de Newton por diferenças finitas, a sua interpretação computacional sobre imagens digitais, a quebra de tal método em algoritmos e por fim os resultados obtidos de sua implementação.

2. FUNDAMENTAÇÃO TEÓRICA

Interpolação é um tipo de função utilizada para encontrar valores entre pontos conhecidos. Existem diversos métodos de interpolação e entre os mais estudados, encontram-se o de Lagrange e o de Newton. A interpolação de Lagrange, segundo [Dav75], possui uma fraqueza visível que pode ser percebida ao -passar de uma dimensão n para outra maior m, pois é necessário determinar um outro conjunto de elementos, tal fraqueza não existe na interpolação de Newton.

2.1 Interpolação de Newton

O enunciado da interpolação de newton por tabela diferenças finitas é que dado um conjunto de pontos 2.1, onde todo x_i possui um valor correspondente y_i, pode-se determinar o valor de um ponto não conhecido 2.2 através da fórmula 2.3, onde h é uma constante que indica a distância de um ponto ao outro.

$$\forall x_i.(x_i \in C\{x_0, x_1, x_2, ..., x_{n-1}\}), \exists y_i.(P(x_i) = y_i) \tag{2.1}$$

$$\forall x_k.(x_k \in [x_0; x_{n-1}])| \tag{2.2}$$

$$P(x) = y_0 + (x - x_0) * \frac{\Delta y_0}{1! * h} + (x - x_0) * (x - x_1) * \frac{\Delta^2 y_0}{2! * h^2} + ... * (x - x_{n-2}) * \frac{\Delta^{n-1} y_0}{(n-1)! * h^{n-1}} \tag{2.3}$$

$$\forall i.(i \in [1; n-1]|x_i - x_{i-1} = h) \tag{2.4}$$

Tabela 2.1 – Tabela de diferenças finitas

x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$
x_0	y_0	$y_1 - y_0$	$\Delta y_1 - \Delta y_0$	$\Delta^2 y_1 - \Delta^2 y_0$
x_1	y_1	$y_2 - y_1$	$\Delta y_2 - \Delta y_1$	—
x_2	y_2	$y_3 - y_2$	—	—
x_3	y_3	—	—	—

Note que, como h é uma constante, é necessário que respeite 2.4. O $\Delta^{\alpha} y_0$ é adquirido calculando-se a tabela de diferenças finitas 2.1. Observa-se que como temos esta tabela, precisamos calculá-la apenas uma vez para que possamos interpolar quantos pontos forem necessários

2.2 Imagens Digitais

Diariamente visualizamos diversas imagens em nossos dispositivos, as quais estão armazenadas em diferentes formatos de arquivos.Cada formato possui um modo diferente de armazenar as informações da imagem,mas em suma uma imagem digital é uma matriz que contém pontos com informações chamados de pixel. Dependendo de como a imagem é armazenada, a informação contida no pixel pode ser apenas um tom de cinza armazenado em 1 byte, ou seja, um valor entre 0 a 255, como visto na figura 2.1 ou pode conter mais valores no caso de uma imagem colorida como a figura 2.2.

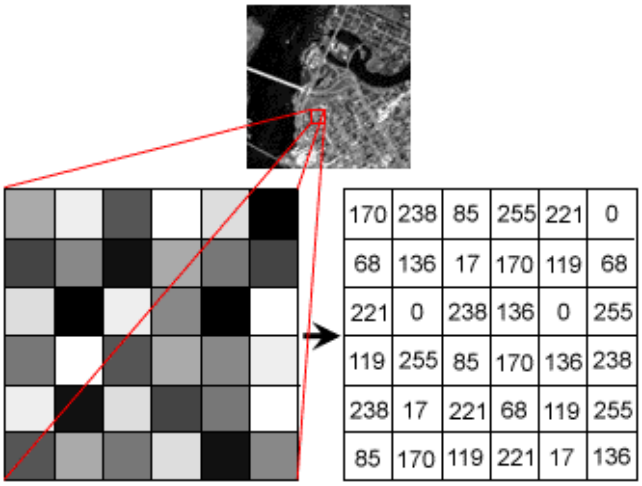

Figura 2.1 – Esta figura representa uma imagem em tons de cinza, já que só existe um valor em cada posição.

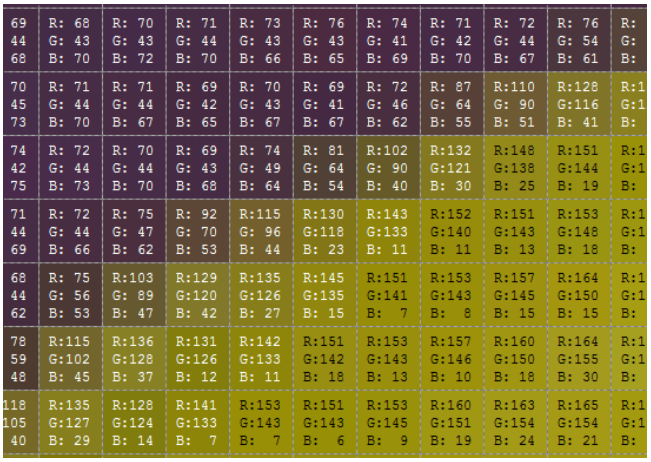

Figura 2.2 – Observa-se que se trata de uma imagem colorida, pois existem 3 valores em cada posição sendo eles respectivamente vermelho, verde e azul

Com o conhecimento de que uma imagem digital é na verdade uma matriz que contem dados, o primeiro passo que precisamos realizar antes mesmo de interpolar a imagem é, caso seja colorida, transformá-la em grayscale. A transformação da imagem em

grayscale é simples, existem diversas fórmulas como por exemplo somar os três valores coloridos como visto na figura 2.2 e dividí-los por três. Neste trabalho, foi utilizada a fórmula 2.5 vista em [Poy98], que leva em consideração a visão humana em preferência do verde.

$$Y = 0.2126R + 0.7152G + 0.0722B \tag{2.5}$$

2.3 Interpolação de Imagens

Agora que já foi introduzido o polinômio de interpolação de Newton proposto neste trabalho e uma breve introdução de o que é uma imagem digital, podemos trazer o contexto de Interpolação para o zoom de imagens.

A interpolação de newton originalmente funciona apenas em uma dimensão,porém é possível utilizá-la em duas dimensões. O primeiro passo é perceber que uma matriz na verdade é um conjunto de vetores um em cima do outro e portanto precisamos gerar uma tabela de diferenças finitas para cada linha da matriz e tal é o algoritmo2.1.

```
1: função calculaTabelasFinitas(M)
2: {Matriz M[y][x]onde x é a largura e y a altura}
3: Tabelas tabelas[y]
4: for i = 0 to y-1 do
5:     calcule a tabela de diferenças finitas de M[i] e armazene em tabelas[i]
6: end for
7: return tabelas
```

Algoritmo 2.1 – Este algoritmo demonstra como gerar tabelas finitas a partir de uma matriz.

Um ponto importantíssimo que não pode passar despercebido é cada em tabela que o algoritmo 2.1 gerou, cada x_i corresponde ao índice da de sua respectiva coluna para a linha que foi utilizada para construir a tabela, o que torna possível utilizar diferenças finitas pelo h sempre 1 e os y_i são as os valores presentes naquela linha e naquele índice.

Após calculadas as tabelas de cada linha, podemos interpolar a matriz como no algoritmo2.2.

Como já sabemos que uma imagem pode ser interpretada como uma matriz, podemos realizar a mesma operação do algoritmo 2.2 em uma imagem para realizar a operação de zoom.

```
1: função interp2D(M, p, tabelas[y], grau)
2: {Matriz M[y][x]onde x é a largura e y a altura}
3: {Lista tabelas[y]lista de mesmo tamanho da altura de M}
4: {Porcentagem p que deseja como matriz resultante}
5: {Grau do polinomioPrecisamos de grau+1 pontos para realizar a interpolação}
6: wx ← x * p * 0.01
7: wy ← y * p * 0.01
8: divy ← wy/y
9: divx ← wx/x
10: Matriz W[wy][wx]
11: for i = 0 to wy-1 do
12:    yMapeado ← (i + 1/divy) − 1
13:    vizinhosLinha[grau+1] ← indiceMaisProximos(M, grau + 1, x)
14:    for j = 0 to wx-1 do
15:       xMapeado ← (j + 1/divx) − 1
16:       vizinhosColuna[grau+1] ← indiceMaisProximos(M, grau + 1, x)
17:       linhasInterpoladas[grau+1]
18:       for k = 0 to grau do
19:          linhasInterpoladas[k] ← Interpolacao1D(tabelas, vizinhosLinha, , xMapeado)
20:       end for
21:       tabelaLinhas[y] ← calculaTabelaFinita(linhasInterpoladas).
22:       W[i][j] ← Interpolacao1D(tabelasLinhas, vizonhosColuna, yMapeado).
23:    end for
24: end for
25: return W
```

Algoritmo 2.2 – Este algoritmo representa como realizar a interpolação 2D em uma dada matriz.

2.4 Caso de Teste

Realizaremos agora um caso de teste sobre os algoritmos da seção anterior. Para simplificar, iremos utilizar uma figura 4x4, que para propósitos didáticos, foi ampliada apenas replicando seus pixeis, algoritmo conhecido como nearest neighbor, como visto na figura 2.3.

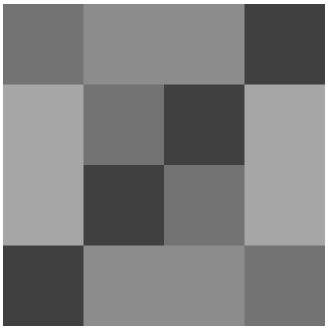

Figura 2.3 – Esta imagem 4x4 é utilizada para demonstrar como funciona o algoritmo.

Desejamos verificar se dada tal imagem como entrada, é possível devolver a mesma saída passando pelo algorítmo. Primeiro verificamos na tabela 2.2 quais são os valores de seus pixels.

Tabela 2.2 – Tabela de Pixels

115	140	140	64
166	115	64	166
166	64	115	166
64	140	140	115

Para cada linha desta tabela, iremos criar uma tabela de diferenças finitas. A linha 1 está na tabela 2.3, a segunda na tabela 2.4, a terceira em 2.5 e por fim a quarta em 2.6.

Tabela 2.3 – Tabela que demonstra a Linha 1

x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$
1	115	25	-25	-51
2	140	0	-76	—
3	140	-76	—	—
4	64	—	—	—

Tabela 2.4 – Tabela que demonstra a Linha 2

x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$
1	166	-51	0	153
2	115	-51	153	—
3	64	102	—	—
4	166	—	—	—

Tabela 2.5 – Tabela que demonstra a Linha 3

x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$
1	166	-102	153	-153
2	64	51	0	—
3	115	51	—	—
4	166	—	—	—

Após realizada a construção destas tabelas, podemos utilizar o algoritmo 2.2 para calcular os pontos de interpolação utilizamos a fórmula 2.3 vista anteriormente. Após o fim do algoritmo, caso o tamanho final seja o mesmo, a tabela será igual à tabela 2.2, consequentemente a imagem resultante será a figura 2.3. É possível também que hajam erros de arredondamento, dependendo do tamanho que se deseja aumentar. Um erro

Tabela 2.6 – Tabela que demonstra a Linha 4

x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$
1	64	76	-76	51
2	140	0	-25	—
3	140	-25	—	—
4	115	—	—	—

necessário de ser tratado é que pode ocorrer do valor final de um ponto da tabela ser maior que 255 ou menor que 0. Caso tal problema ocorra deve-se definí-los como o limite correspondente.

3. IMPLEMENTAÇÃO

A implementação deste trabalho foi realizada sobre a linguagem de C++. A escolha de uma linguagem que possibilita a programação de baixo nível veio a partir do conhecimento que a computação inicial da tabela de diferenças finitas de Newton é computacionalmente custosa não só pelo número de operações, mas como a quantidade de dados que em parte utiliza amplamente e em outra fica ocupando a memória. Logo, o melhor que se possa fazer é utilizar tal linguagem com estruturas que possibilitem um amplo controle de memória. Uma importante otimização que pode-se observar é que como h é 1, podemos desconsiderar ele no cálculo da fórmula 2.3. Com propósito de simplificar o uso do programa, uma interface foi criada como visto na figura 3.1. A figura3.2 demonstra como funciona os resultados adquiridos pela operação de zoom.

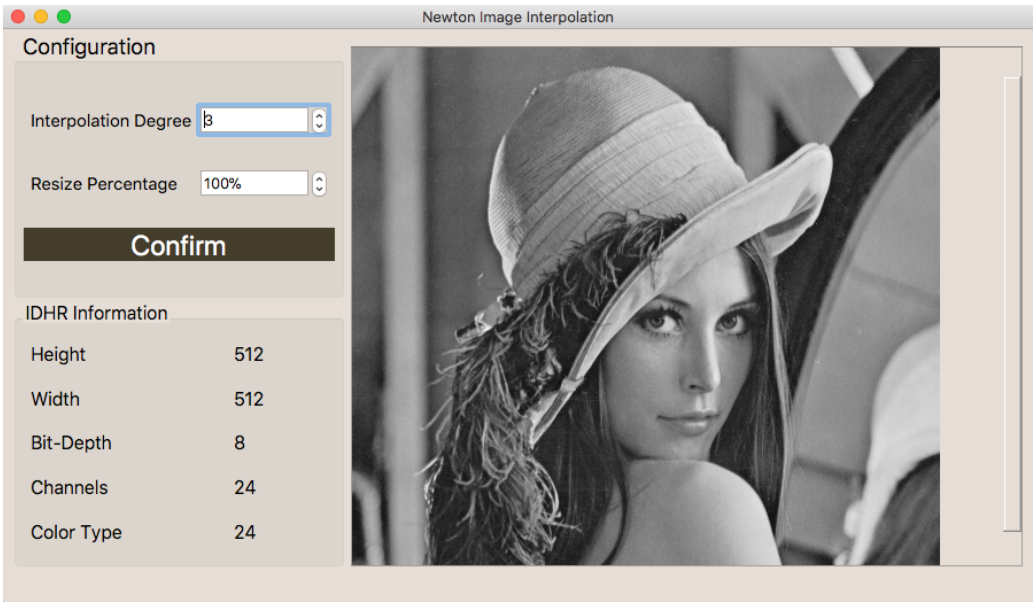

Figura 3.1 – Interace de usuario para aplicar o zoom.

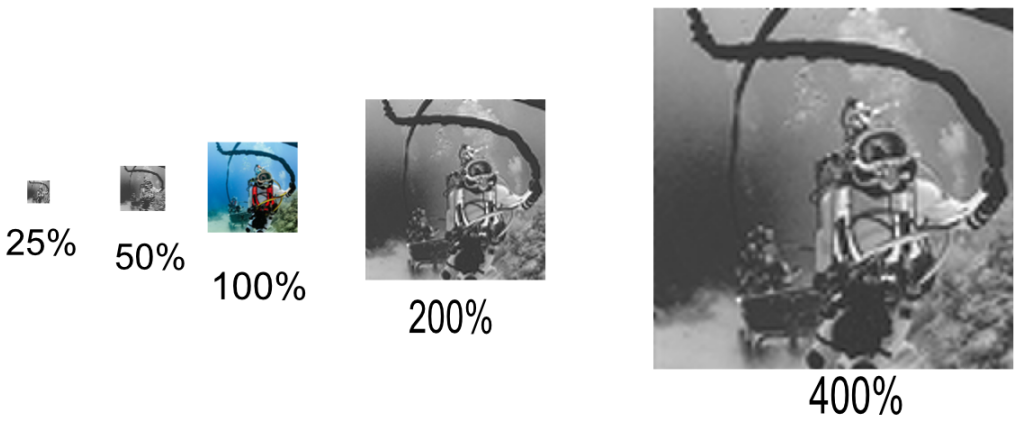

Figura 3.2 – Pode-se observar a partir dessa imagem as operações de zoom-in e zoom-out.

As operações realizadas na 3.2 utilizaram o polinômio de newton com o 3 grau, ou seja, com os 4 pontos mais próximos. Um problema considerável de se aumentar tal tamanho pode ser visto na figura 3.3 onde houve a tentativa de aumentar a mesma figura utilizando o grau 25 , com 26 pontos mais próximos.

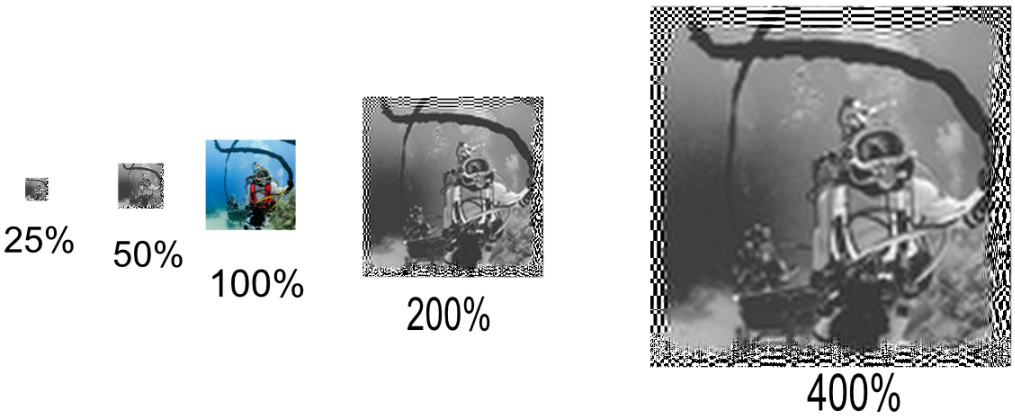

Figura 3.3 – Pode-se nesta operação de zoom estão com erros.

4. CONCLUSÃO

Embora o método de Newton tenha dado resultados relativamente satisfatórios se utilizados com polinômios de grau baixo, ele ainda apresenta certos problemas como por exemplo um grande erro de arredondamento causado por sucessivas operações como visto na figura 3.3 caso seja utilizado sobre todos os elementos da linha e depois sobre todas as colunas, como o método original determina, de fato o resultado não seria satisfatório. Por este motivo optou-se por implementar uma opção de escolha de grau do polinômio, visando salientar este problema. Na implementação utilizou-se dados do tipo *float* na hora de realizar as operações e isso de fato afetou os cálculos, porém mesmo que utilizássemos o tipo *longdouble*, em algum o mesmo erro ocorreria, e além disso o próprio programa ficaria muito mais lento.

REFERÊNCIAS BIBLIOGRÁFICAS

[Dav75] Davis, P. J. “Interpolation & Approximation”. New York, NY: Dover Publications, INC, 1975, 39p.

[Poy98] Poynton, C. A. “Rehabilitation of gamma”. In: Human Vision and Electronic Imaging III, San Jose, CA, USA, January 24, 1998, 1998, pp. 232–251.