数字图象放大[1]

摘要

随着现代成像设备的不断进步和网络通信技术的飞速发展，数字图像的应用越来越普及。数字图像处理技术逐渐成为信息处理领域的研究热点，得到了快速的发展。图像放大是图像处理和应用中的一种常见的操作，也是一些其他更复杂处理的基础，其主要目的是提高放大后图像的分辨率，以满足人们的视觉享受或实际应用要求。在视频安全监控、遥感成像、高清电视、医学图像等领域，图像放大技术都有重要应用。

现在的数字图像发达算法几乎都是采用内插值算法，即在原有图像像素的基础上在像素点之间采用合适的插值算法插入新的元素。本文主要通过建立模型研究几种常见的图像插值算法(双线性插值算法、双三次插值算法和三次样条插值算法)在插值质量和时间上的差异。

问题重述

图像放大过程中，若使用一些较为简单的算法，虽然速度较快，但往往存在色彩失真，图像的细节无法得到较好的呈现等问题。若使用一些较为复杂的算法，虽然能得到较好的效果，但往往因为复杂度过高，导致所需计算量过大，时间过长，不具备实时性。

本文通过分析几种常见的图像插值算法，比较其插值质量，所需时间的差异。旨在找出一种合适的插值算法，在放大后的图像质量与放大过程的消耗时间之间取得平衡。

问题分析

图像放大有许多算法，其关键在于对未知像素使用何种插值方式。针对图像放大的质量，可以通过将图像缩小后，重新放大至原始大小。然后通过比较新放大的图像是否存在色彩失真，图像的细节是否的得到较好的呈现，计算新放大出的图像与原始图像的差异，从而得出图像放大的质量。

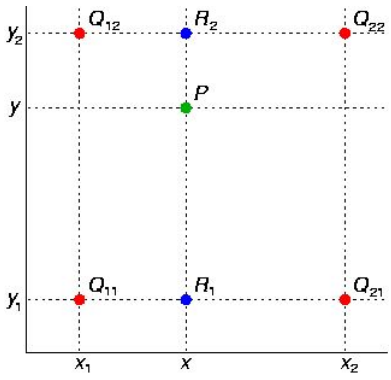
针对图像放大的时间消耗，可以通过放大实际所需时间得出，比较各种放大算法所用时间。

当把一个小图像放大时，对于放大后图像中的每一个像素点，我们可以先通过坐标变换，得到其在被放大图像中的位置。然后根据其周围像素点的颜色值，通过插值算法，对其进行估计。

本文主要考虑一下三种插值算法：

1. 双线性插值算法

双线性插值，又称为双线性内插。在数学上，双线性插值是对有两个变量的插值函数的线性插值扩展，其核心思想是在两个方向分别进行一次线性插值。



设放大后图像中的一个像素点在被放大图像中对应的像素点 $P=(x,y)$，假设已知其临近的整数像素点 $Q_{11}=(x_1,y_1)$， $Q_{12}=(x_1,y_2)$， $Q_{21}=(x_2,y_1)$ 及 $Q_{22}=(x_2,y_2)$ 四个点的颜色值。

首先在 x 方向进行线性插值，得到

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

然后在 y 方向进行线性插值，得到

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

这样就得到所要的结果 $f(x, y)$

$$\begin{aligned} f(x,y) \approx & \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y_2 - y) + \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y_2 - y) \\ & + \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y - y_1) + \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y - y_1) \end{aligned}$$

双线性插值算法由于插值结果是连续的，所以视觉上会比最临近点插值算法要好一些，不过运算速度稍微慢一点，如果讲究速度，是一个不错的折衷。

2. 双三次插值算法（双立方插值）

双立方插值算法与双线性插值算法类似，对于放大后未知的像素点 P，将对其有影响的范围扩大到邻近的 16 个像素点，依据对 P 的远近影响来进行插值。因 P 点的像素值信息来自 16 个邻近点，所以可得到较细腻的影像，不过速度较慢。

设放大后图像中的一个像素点在被放大图像中对应的像素点 $P = (i + u, j + v)$，其中 i、j 均为非负整数，u、v 为 $[0,1)$ 区间的浮点数。则像素值 $f(i + u, j + v)$ 可由

如下插值公式得到：

$$f(i+u,j+v)=[A]*[B]*[C]$$

$$[A]=[S(u+1)\quad S(u+0)\quad S(u-1)\quad S(u-2)]$$

$$[B]=\begin{bmatrix} f(i-1,j-1) & f(i-1,j+0) & f(i-1,j+1) & f(i-1,j+2) \\ f(i+0,j-1) & f(i+0,j+0) & f(i+0,j+1) & f(i+0,j+2) \\ f(i+1,j-1) & f(i+1,j+0) & f(i+1,j+1) & f(i+1,j+2) \\ f(i+2,j-1) & f(i+2,j+0) & f(i+2,j+1) & f(i+2,j+2) \end{bmatrix}$$

$$[C]=\begin{bmatrix} S(v+1) \\ S(v+0) \\ S(v-1) \\ S(v-2) \end{bmatrix}$$

其中插值基函数 $S(x)$ 为

$$S(x)=\begin{cases} 1-2|x|^2+|x|^3 & ,0\le|x|<1 \\ 4-8|x|+5|x|^2-|x|^3 & ,1\le|x|<2 \\ 0 & ,|x|\ge 2 \end{cases}$$

3. 三次样条插值算法

三次样条插值算法主要通过三次多项式在各像素点之间构建一个三次样条，通过低阶多项式样条实现较小的插值误差。

模型假设

设原始图像[2]A 大小为 $M*N$ 。

缩小后图像大小为 $\frac{M}{k}*\frac{N}{k}$ 。

则重新放大后图像 C 大小为 $M*N$ 。

为简便起见，本文选取放大倍数 $k=2$ 。

本文选取待放大图像为 1a. jpg、2a. jpg、3a. jpg、4a. jpg 和 5a. jpg

[2] 本文所讨论图像统一采用每像素 24 位编码的 RGB 值表示:
使用三个 8 位无符号整数(0 到 255)表示红色(R)、绿色(G)和蓝色(B)的强度。

符号说明

A	原始图像
B	缩小后图像
C	重新放大后图像
K	放大系数
$M*N$	图像 A 及图像 C 大小，M 行 N 列
$\frac{M}{k}*\frac{N}{k}$	图像 B 大小，$\frac{M}{k}$ 行 $\frac{N}{k}$ 列
$f(x,y)$	像素点 (x,y) 的颜色值，$f(x,y)=(R(x,y),G(x,y),B(x,y))$
s^2	定义的误差估计值
$t(n)$	进行 n 次图像放大所需时间

模型的建立及求解

1、放大质量

重新放大后的图形与原始图像的差值由如下公式给出

$$s^2=\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(R_A(i,j)-R_C(i,j))^2+(G_A(i,j)-G_C(i,j))^2+(B_A(i,j)-B_C(i,j))^2}{3MN}$$

此公式依次计算图形 A 与图形 C 中对应点像素值的 R、G、B 三个分量的绝对偏差，为了避免负值的影响，故平方后求和。最后除以总点数，再除以每个像素点的颜色分量数 3，即可用来估计每个分量的偏差情况。

本文选取 5 张图像，通过 MATLAB，分别使用双线性插值算法、双三次插值算法和三次样条插值算法进行放大，得到结果如下

图像名称	1a.jpg	2a.jpg	3a.jpg	4a.jpg	5a.jpg
图像大小	480 * 320	190 * 300	300 * 241	280 * 358	325 * 325
放大倍数	k = 2				
双线性插值 s^2	495.0329	96.8998	133.6443	414.5436	154.3460
双三次插值 s^2	481.7380	90.5473	132.4180	407.7585	147.6325
三次样条插值 s^2	484.5879	88.7960	134.4205	408.2020	145.7144

由以上统计信息可以看出，对于同一图像，不同的插值算法放大后的图像误差总体较为接近，但具体比较，双三次插值算法和三次样条插值算法要明显好于双线性插值的结果，而双三次插值和三次样条插值面对不同的图像各有优缺。

而就不同的图像而言，即使采用同一插值算法，得到的结果也存在较大差异。此

差异的来源主要取决于待放大图像的信息，特别是其细节突变的数量。

如图像 1a. jpg 和 4a. jpg，图中主要内容为树，包含了大量的树叶，图像所呈现的细节突变较多，而插值依赖周围数个点的像素值，无法较好的反映出图像细节突变，故插值后误差较大。

而图像 2a. jpg、3a. jpg 和 5a. jpg，图像主要内容为风景，图像呈现出明显的渐变特性，细节突变较少，故插值后误差较小。

2、放大所需时间

放大操作所需时间通过使用 MATLAB 对图像进行放大操作，统计耗时长短得出。

为了准确，统计 1000 次放大操作所用时间。同时，考虑到 IO 操作的瓶颈，为了避免 IO 操作的影响，只对放大操作所耗 CPU 时间进行统计，本文测试所用 CPU 为 Intel I5-3230。

图像名称	1a.jpg	2a.jpg	3a.jpg	4a.jpg	5a.jpg
图像大小	480 * 320	190 * 300	300 * 241	280 * 358	325 * 325
放大倍数	k = 2				
双线性插值 $t(1000)$	144.7813	58.5469	70.4063	93.8750	100.5313
双三次插值 $t(1000)$	201.0938	77.7813	95.8594	130.0625	133.9531
三次样条插值 $t(1000)$	270.7500	102.8438	127.0781	169.0625	173.2188

由以上统计信息可以看出，不同的插值算法所消耗的时间存在明显差异。双三次插值算法的时间消耗较双线性插值算法越多 35%，三次样条插值算法的时间消耗较双线性插值算法越多 70%。

故在进行图像放大时候必须考虑要所需要消耗的时间，考虑到能否在规定时间内完成相应的图片放大操作。

3、改进算法

考虑到以上结论，可以对图像的放大提出如下改进方案：

1、先对一张图像整体进行扫描，将图像进行分区。分区依据为该部分内相邻像素点像素值的突变程度。

2、对图像中相邻像素点突变程度较小的区域，说明这一部分图像具有明显的渐变特性，像素值变换较小，相邻像素值间具有明显的相关关系。可以使用双线性插值算法对这一部分进行放大，这样，可以再不损失放大质量的前提下，获得较好的图像质量。

3、对图像中相邻像素点突变程度较大的区域，说明这一部分图像存在细节的变换，相邻像素值间关系较小。可以采用双三次插值算法对这一区域进行放大，以获得更为精确的插值结果。

采用此方案，对比前文所提三种算法，可以在插值质量和时间消耗之间获得较好的平衡。在较短的时间内达到不错的插值质量。

模型的优缺点

1、优点

通过比较重新放大后图像各个插值像素点的值与原始图像的差异，较好的刻画了插值的质量。

通过多次插值，统计出了各种插值算法所消耗的时间。

2、缺点

模型只考虑了整体误差，没有反映出插值后图像个别点的误差情况。

参考文献

[1] 维基百科 双线性插值

https://zh.wikipedia.org/wiki/%E5%8F%8C%E7%BA%BF%E6%80%A7%E6%8F%92%E5%80%BC

[2] 维基百科 双三方插值

https://zh.wikipedia.org/wiki/%E5%8F%8C%E4%B8%89%E6%AC%A1%E6%8F%92%E5%80%BC

[3] 维基百科 样条插值

https://zh.wikipedia.org/wiki/%E6%A0%B7%E6%9D%A1%E6%8F%92%E5%80%BC

[4] 王会鹏, 周利莉, 张杰. 一种基于区域的双三次图像插值算法[J]. 计算机工程, 2010, 36(19): 216-218.

[5] 易珺. 图像放大方法概述[J]. 电视字幕. 特技与动画, 2008, 14(7): 35-37.

[6] 李将云. 图像处理中的插值和缩放若干技术研究[D]. 杭州: 浙江大学, 2002.

附录

```
% fun.m
% 图像放大及误差统计

clear;
close all;

src = imread('1a.jpg'); % 待放大图像
source = imread('1.jpg'); % 原始图像，用于评估结果
k = 2; % 放大倍数
method = 'linear';

% nearest: 最近邻插值法
% linear: 线性插值法
% cubic: 三次多项式插值法
% spline: 三次样条插值法

% 图像放大
src = double(src);
m = size(src, 1);
n = size(src, 2);
[x, y] = meshgrid(1 : n, 1 : m);
[xi, yi] = meshgrid(1 / k : 1 / k : n, 1 / k : 1 / k : m);
dst( : , : , 1) = interp2(x, y, src( : , : , 1), xi, yi, method);
dst( : , : , 2) = interp2(x, y, src( : , : , 2), xi, yi, method);
dst( : , : , 3) = interp2(x, y, src( : , : , 3), xi, yi, method);

imshow(uint8(dst));

% 误差统计
s1 = double(0);
totalNum = 0; % 记录总点数
for i = 1 : 1 : min(size(source, 1), size(dst, 1))
    for j = 1 : 1 : min(size(source, 2), size(dst, 1))
        totalNum = totalNum + 1;
        for q = 1 : 1 : 3
            s1 = s1 + double(source(i, j, q) - dst(i, j, q)) ^ 2;
        end
    end
end
s1 = s1 / (3 * totalNum);
disp(s1);
```

```
% time.m
% 统计 1000 次放大操作所需时间

clear;
close all;

src = imread('1a.jpg'); % 待放大图像
source = imread('1.jpg'); % 原始图像，用于评估结果
k = 2; % 放大倍数
method = 'linear';

% nearest：最近邻插值法
% linear：线性插值法
% cubic：三次多项式插值法
% spline：三次样条插值法

% 图像放大
src = double(src);
m = size(src, 1);
n = size(src, 2);

tic
t0 = cputime()
for i = 1 : 1000
    [x, y] = meshgrid(1 : n, 1 : m);
    [xi, yi] = meshgrid(1 / k : 1 / k : n, 1 / k : 1 / k : m);
    dst( : , : , 1) = interp2(x, y, src( : , : , 1), xi, yi, method);
    dst( : , : , 2) = interp2(x, y, src( : , : , 2), xi, yi, method);
    dst( : , : , 3) = interp2(x, y, src( : , : , 3), xi, yi, method);
end
t1 = cputime()
toc
disp(t1 - t0)

% imshow(uint8(dst));
```