Image Processing
Home work 01
Different Types of Interpolation

Student: Aqeel Labash
Teacher: Gholamreza Anbarjafari

17 February 2016

1 The Inverse Distance to a Power Method

As I understood it's the same of Shepard's method or Inverse Distance weighting[1]. Simply the more distance between new point and the points around it the less effective it will be on the new point. It's given by following relation:

$$u(x) = \begin{cases} \frac{\sum_{i=1}^{N} w_i(x_i)u_i}{\sum_{i=1}^{N} w_i(x_i)}, & if\ d(x,x_i) \neq 0 \\ u_i, & if\ d(x,x_i) = 0 \end{cases}$$

$$u_i = u(x_i) \text{ for } i = 1, 2...N$$

$$w_i(x) = \frac{1}{d(x,x_i)^p}$$

From the previous formulas we notice that weight represent the influence this value on the new value. how much we are going to take from that neighbor.

2 The Kriging Method

The Kriging method also known as Gaussian processes is non-linear interpolation tool [3].Kriging method estimate point based on observed values (in our case pixels surrounding the new pixel) and it's defined by formula [2]:

$$\hat{Z}_0 = \sum_{i=1}^{N} w_i Zi \text{ where } \sum w_i = 1$$

The weights depend on the spatial covariance values.The advantages of Kirging [4]

1. Treat clusters as single point.
2. Estimate of estimation error(Kriging variance.)
3. "provide basis for stochastic simulation of possible realization of $Z(u)$. [4]

3 The Minimum Curvature Method

This method is mainly used in earth science [1].It's "analgous to a thin,linearly elastic plate passing through each of the data values with a minimum amount of bending." [5] It's also based on the modified biharmonic differential equation.

4 The Modified Shepard's Method

The modified Shepard's method differ from Shepard's method by using nearest neighbors in R-sphere[3].The weight is given by the following formula:

$$w_k(x) = \left(\frac{max(0, R - d(x,x_k))}{Rd(x,x_k)} \right)^2$$

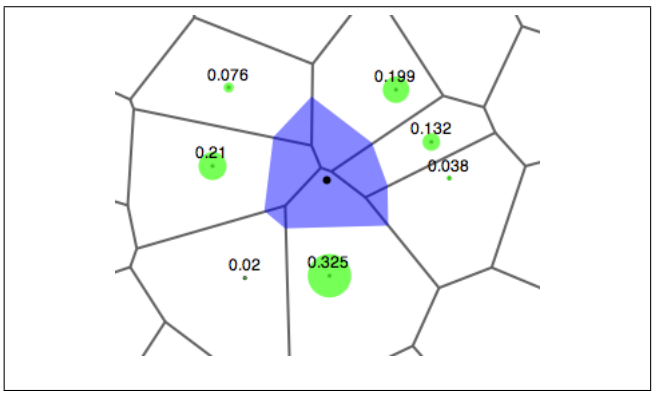
The previous formula allow to reduce the bull's-eye apperance of the generated contours.[1][3]

5 The Natural Neighbor Method

This method advantage is that it's provide smoother approximation and it use the following formula :

$$G(x,y) = \sum_{i=1}^{n} w_i f(x_i, y_i)$$

where w_i by "finding how much of each of the surrounding areas is "stolen" when inserting (x,y) into the tessellation."[6] where the values $(1,n)$ represent the surrounding area and $f(x_i,y_i)$ represent specific area value.



6 The Nearest Neighbor Method

In this method we assign the value of the nearest neighbor to the new pixel.This method useful when data evenly spaced.It could be used effectively if there isn't much points to be generated other wise it'll be problematic.[1]

7 The Polynomial Regression Method

This method try to find the patterns in the data and then predict the new point.it's general formula is :

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_n x^n + \epsilon$$

after figuring out $a_0, a_1, .., a_n$ we can predict the new points.

8 The Radial Basis Function Interpolation Method

The value of this method depends only on the distance from the new point or pixel.[7] There is many types for this method (Gaussian, multiquadric, inverse quadric, inverse multiquadric, polyharmonic spline,thin plate spline). It's approximation is given by the following formula:

$$y(x) = \sum_{i=1}^{N} w_i \phi(||x - x_i||)$$

The multiquadrtic type is considered to be the best [1] in producing smooth value.Which is considered by this formula :

$$\phi(r) = \sqrt{1 + (\epsilon r)^2}, r = ||x - x_i||$$

The weights estimated by linear least squares due to the linearity of weights in the approximation function.[7]

9 The Triangulation Linear Interpolation Method

This method connect the points with triangles where no two share the same edge.This way any point would be identified by the three points of the triangle it belongs to. Best result from this method come out when the data are evenly distributed.[1]

10 The Moving Average Method

This method define a search ellipse with minimum number of data to use.To identify a point the ellipse will be centered at that point.The point value will be the average of all data points within ellipse.The ellipse should satisfy the minimum required number of data points other wise the value would be blanked.[1]

11 The Data Metrics Method

In this method we obtain the following information for each node :

1. number of point to interpolate.
2. Standard deviation, variance, coefficient of variation,median absolute deviation.
3. distance to the nearest point.

So to get values for new point we use the nearest data metric to that point to calculate the new value.

12 The Local Polynomial Method

This method use weighted least squared fit for data within the search ellipse to get the new value.It's given by the following formula for 2d pixel:

$$I(x, y) = \sum_{i,j=0}^{n} c_{ij} x^i y^i$$

coefficient calculated by using the least square fit .[8] In this method the pixel is processed in blocks (3X3,4X4 etc..). This method can be used for image down-sampling, grey scale image compression,noise filtration.[8]

References

[1] Chin-Shung Yang* Szu-Pyng Kao* Fen-Bin Lee** Pen-Shang Hung** ,TWELVE DIFFERENT INTERPOLATION METHODS:A CASE STUDY OF SURFER 8.0

[2] Kriging Example

[3] Interpolation - Wikipedia

[4] Krigin

[5] Mgr. Miroslav Dressler,Art of Surface Interpolation

[6] Sibson, R. (1981). "A brief description of natural neighbor interpolation (Chapter 2)". In V. Barnett. Interpreting Multivariate Data. Chichester: John Wiley. pp. 21–36

[7] Radial basis function approximation

[8] Guennadi Levkine , In Vancouver, Canada, Image Local Polynomial Approximation (LPA) and it's Applications

PROGRAMMING PART

Lanczos Function

Lanczos kernel is given by the following formula:

$$L(x) = \begin{cases} sinc(x)sinc(\frac{x}{a}), & \text{if } -a < x < a \\ 0 & \text{other wise} \end{cases}$$

It's also can be written:

$$L(x) = \begin{cases} 1 & \text{if } x = 0 \\ \frac{asin(\pi x)sin(\frac{\pi x}{a})}{\pi^2 x^2} & \text{if } 0 < |x| < a \\ 0 & \text{other wise} \end{cases}$$

Methods Comparisons

Before Start

The Obelix image resolution is (2336,2724). My picture resolution (429,592). I decreased there resolution to (200,ratio). Here is the code for changing the resolution (considering aspect ratio) :

```
obeleximg = cv2.imread("Obelix.jpg")
r = float(200)/float(obeleximg.shape[1])
dim = (200,int(obeleximg.shape[0]*r))
obeleximg = cv2.resize(obeleximg,dim)
cv2.imwrite("Obelexorignalgrayscal.png",obeleximg)

mypicture = cv2.imread("mypicture.JPG")
r = float(200)/float(mypicture.shape[1])
dim = (200,int(mypicture.shape[0]*r))
print dim
mypicture = cv2.resize(mypicture,dim)
cv2.imwrite("mupictureorignalgrayscal.png",mypicture)
```

Performance Comparisons

After that I used python with opencv to double the images size for (Nearst,bilinear,bicubic) and used matlab for (lanczos ,spline).
The performance was as following Table measured in microseconds :

Picture	lanczos4	Nearest	bilinear	bicubic	spline
Obelex	3951	375	508	849	100000
Me	1608	193	260	505	120000

And here is the code used to generate the pictures :

```
import cv2
from time import time

def InterpolateImage(img,way,filename):
  start = time()
  resized = cv2.resize(img,(img.shape[1]*2,img.shape[0]*2),interpolation=way)
  print filename+" : ", (time()-start)*1000000
  cv2.imwrite(filename+".png",resized)

InterpolateImage(obeleximg,cv2.INTER_NEAREST,'ObelexNearst')
InterpolateImage(obeleximg,cv2.INTER_LINEAR,'obelexBilinear')
InterpolateImage(obeleximg,cv2.INTER_CUBIC,'ObelexCubic')
InterpolateImage(obeleximg,cv2.INTER_LANCZOS4,'ObelexLanczos')

InterpolateImage(mypicture,cv2.INTER_NEAREST,'mypictureNearst')
InterpolateImage(mypicture,cv2.INTER_LINEAR,'mypictureBilinear')
InterpolateImage(mypicture,cv2.INTER_CUBIC,'mypictureCubic')
InterpolateImage(mypicture,cv2.INTER_LANCZOS4,'mypictureLanczos')
```

Images Comparison

Figure 1: Obelex original

This is the original image after resizing.

Figure 2: Obelex nearest neighbor

In this image we can notice the squares every where.

Figure 3: obelex Bilinear.png

In figure 3 we can notice that squares are less than figure 2 but still more blur

Figure 4: Obelex Cubic

In figure 4 there still some squares but less blur than Bilinear (figure 3)

Figure 5: Obelix Spline

Here it has the same problems in figure 4 we still have squares but less than the previous techniques

Figure 6: Obelex Lanczos

Lanzos : so far the best one , least squares , least blur.
The implementation for Lanczos on Matlap is here ,main.m file :

```
obilixpicture = imread('Obelexorignalgrayscal.png');
mypicture=imread('mupictureorignalgrayscal.png');

obi_large = imresize(obilixpicture,2,{@lanczos4,4});
mypicturelarge=imresize(mypicture,2,{@lanczos4,4});

imwrite(mypicturelarge,'mypictureLanczos')
imwrite(obi_large,'ObelexLanczos')
```

and lanczos4.m

```
function f = lanczos4(x)
f = (sin(pi*x) .* sin(pi*x/4) + eps) ./ ((pi^2 * x.^2 / 4) + eps);
f = f .* (abs(x) <4);
end
```

Note: I used python code to generate the pictures, because I use linux and there was a problem executing the previous matlab code.

Figure 7: My Picture original

My picture after resizing.

Figure 8: My Picture nearest neighbor

We notice squares everywhere here as well.

Figure 9: My Picture Bilinear

For Bilinear here actually some how I feel there is almost no squares but it is blur.

Figure 10: My Picture Cubic

In Cubic we notice that it's less blur than Bilinear and less squares than nearest neighbor.

Figure 11: My Picture Spline

Even less squares specially near the eyebrows.

Figure 12: Lanczos4

Although lanczos is the most expensive one in time but in this example didn't give the result I expect to be honest. Spline got a better result.

Since spline was spoused to be a built-in function I used this copied code to calculate it :[2]

```
%THIS CODE IS COPIED FROM THIS REPOSITORY
%https://www.reddit.com/r/matlab/comments/32vfj0/using_interp2_to_resize_an_imageinstead_of/

cd ('/home/aqeel/Study/IP/hw1/')
%M = imread('mupictureorignalgrayscal.png');
M = imread('Obelexorignalgrayscal.png');
```

```
t = cputime;
M = double(M);

[Y,X,z] = size(M);
MaxD = max(size(M));
%Scale = 550/MaxD; % create the scale to be applied to both dimensions
Scale = 466/MaxD;
[x,y] = meshgrid(1:X,1:Y); % the grid for the original image

[Xq,Yq] = meshgrid(1:Scale*X,1:Scale*Y); % the grid for the new image

MqR = interp2(x,y,M(:,:,1),Xq,Yq,'spline');
MqG = interp2(x,y,M(:,:,2),Xq,Yq,'spline');
MqB = interp2(x,y,M(:,:,3),Xq,Yq,'spline');

Mq = zeros(floor(Scale*Y),floor(Scale*X),3); % initialize new image matrix

Mq(:,:,1) = MqR;
Mq(:,:,2) = MqG;
Mq(:,:,3) = MqB;

Mq = uint8(Mq);

Xi = linspace(1,X,round(X)*Scale);
Yi = linspace(1,Y,round(Y)*Scale);

[Xq,Yq] = meshgrid(Xi,Yi); % **why do I have to swap the X and Y places in the argument??**

Mq = zeros(floor(Y*Scale),floor(X*Scale),3);

for i = 1:3
Mq(:,:,i) = interp2(M(:,:,i),Xq,Yq,'spline');
end

Mq = uint8(Mq);
e = cputime-t
imwrite(Mq,'oblexspline.png')
%imwrite(Mq,'mypicturespline.png')
```

Code Source[2]

Note: All resources for this home work is available at my repository.

References

[1] Wilhelm Burger, Mark J. Burge (2009). Principles of digital image processing: core algorithms. Springer. pp. 231–232. ISBN 978-1-84800-194-7.

[2] Code Source for spline