**Image Processing**

**Project #1 - Spatial Transformation**

**Due date: 2015/10/18**

**0456648 多工碩一 楊柏漢**

**繳交日期 : 2015/10/17**

文件列表:

1. project1\_NN.m

(nearest neighbor interpolation的裁切、縮放、旋轉)

1. project1\_bilinear.m

(bilinear interpolation 的裁切、縮放、旋轉)

1. project1\_bicubic.m

(bicubic interpolation 的裁切、縮放、旋轉)

1. Fig0236(a)(letter\_T).tif

(作業附圖檔)

1. Fig0220(a)(chronometer 3692x2812 2pt25 inch 1250 dpi).tif

(作業附圖檔)

1. project1\_NN.tif

(利用nearest neighbor interpolation縮放5倍後的成像圖)

1. project1\_bilinear.tif

(利用bilinear interpolation縮放5倍後的成像圖)

1. project1\_bicubic.tif

(利用 bicubic interpolation縮放5倍後的成像圖)

1. IP\_Project1\_0456648.doc

(本文件, project report)

**Technical discussion**

本次Project1完成使用的軟體: Matlab，Matlab在進行矩陣運算比起其他程式語言來的快，故使用此軟體來實作。

分別完成的項目為 nearest neighbor interpolation、bilinear interpolation、 bicubic interpolation，以下將分別解說本次project針對各種內插方法放大圖像之程式撰寫

1. Nearest neighbor interpolation

最近鄰差補，指定給每一個新位置在原影像中最鄰近該位置的強度值。

節錄程式碼做解說:

for j=0:height1

for i=0:width1 l\_fig(ceil(1+j\*scale\_h):ceil(1+j\*scale\_h)+scale\_h,ceil(1+i\*scale\_w):ceil(1+i\*scale\_w)+scale\_w) = fig(1+j,1+i);

%利用迴圈將每個點依照放大倍率映射到新的矩陣(l\_fig)中，並補足漏掉的點

end

end

clear fig;

fig = l\_fig;

1. Bilinear interpolation

雙線性補差，使用四個最鄰近的點來估測一個所給位置處的強度，

依照上課講義可以查閱到以下公式:

Z = Z1(1-u)(1-t) + Z2u(1-t) + Z3t(1-u) + Z4ut

節錄程式碼做解說:

for b=1:height2-scale\_h

for a=1:width2-scale\_w

x1 = scale\_h\*(floor((b-1)/scale\_h))+1; %找出最鄰近點之x1

y1 = scale\_w\*(floor((a-1)/scale\_w))+1; %找出最鄰近點之y1

x2 = scale\_h\*(ceil(b/scale\_h))+1; %找出最鄰近點之x2

y2 = scale\_w\*(ceil(a/scale\_w))+1; %找出最鄰近點之y2

z1 = fig(x1, y1); z2 = fig(x1, y2); z3= fig(x2, y1); z4 = fig(x2, y2); %令最相鄰點的點座標z1, z2, z3, z4

u = (a-y1)/scale\_w; t = (b-x1)/scale\_h; %算出u與t

z = z1\*(1-u)\*(1-t) + z2\*u\*(1-t) + z3\*t\*(1-u) + z4\*u\*t; % 帶入 bilinear 公式

fig(b,a)=z; %將公式求得的z取代回原本的點

end

end

1. Bicubic interpolation

雙立方補差，此方法涉及一個點的16個最近鄰，指定給點公式運算(x,y)的強度值

節錄程式碼做解說:

for m = 1:rn

x1 = ceil(m/s); x2 = x1+1; x3 = x2+1;

p = cast(x1,'uint16');

if(s>1)

m1 = ceil(s\*(x1-1)); %找出最近鄰

...

end

X = [ (m-m2)\*(m-m3)\*(m-m4)/((m1-m2)\*(m1-m3)\*(m1-m4)) ...

(m-m1)\*(m-m3)\*(m-m4)/((m2-m1)\*(m2-m3)\*(m2-m4)) ...

(m-m1)\*(m-m2)\*(m-m4)/((m3-m1)\*(m3-m2)\*(m3-m4)) ...

(m-m1)\*(m-m2)\*(m-m3)/((m4-m1)\*(m4-m2)\*(m4-m3))];

%算出4\*4矩陣X

for n = 1:cn

y1 = ceil(n/s); y2 = y1+1; y3 = y2+1;

if (s>1)

n1 = ceil(s\*(y1-1)); %找出最近鄰

...

end

Y = [ (n-n2)\*(n-n3)\*(n-n4)/((n1-n2)\*(n1-n3)\*(n1-n4));...

(n-n1)\*(n-n3)\*(n-n4)/((n2-n1)\*(n2-n3)\*(n2-n4));...

(n-n1)\*(n-n2)\*(n-n4)/((n3-n1)\*(n3-n2)\*(n3-n4));...

(n-n1)\*(n-n2)\*(n-n3)/((n4-n1)\*(n4-n2)\*(n4-n3))];

%算出4\*4矩陣Y

q = cast(y1,'uint16');

sample = im\_pad(p:p+3,q:q+3,:);

im\_zoom(m,n,1) = X\*sample(:,:,1)\*Y; %公式運算矩陣相乘

if(d~=1)

im\_zoom(m,n,2) = X\*sample(:,:,2)\*Y;

im\_zoom(m,n,3) = X\*sample(:,:,3)\*Y;

end

end

end

**Discussion of results**

利用nearest neighbor interpolation縮小5倍在放大為原尺寸繪出的如下:

**C:\Users\Lescure\Documents\MATLAB\project1_NN.tif**

會產生不理想人造物景的傾向，例如上圖中的筆直邊緣的嚴重失真。

利用bilinear interpolation縮小5倍在放大為原尺寸繪出的如下: C:\Users\Lescure\Documents\MATLAB\project1_bilinear.tif

利用bilinear的結果比nearest nighbor產生好很多的結果，但計算負擔略增加

利用bicubic interpolation縮小5倍在放大為原尺寸繪出的如下:

C:\Users\Lescure\Documents\MATLAB\project1_bicubic.tif

Bicubic interpolation在保存細節上比bilinear interpolation來的好，但相對花費的計算時間又更久。

程式碼中還含有”裁切”、”縮小”、”旋轉”、”水平位移”等實作應用，於程式碼中另以註解方式呈現，謝謝。