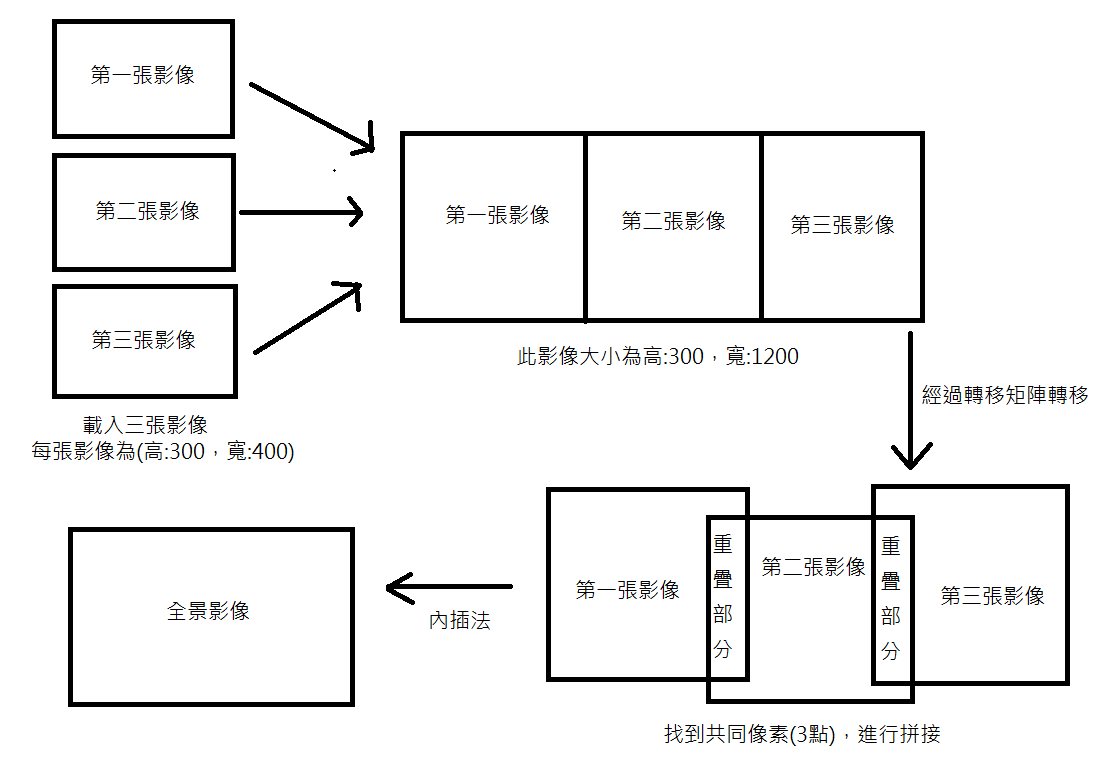
7108056022 陳冠霖

作法:



第一步:

讀入三張需要拼接的影像。

程式碼:

/\*按下button1(import1)載入拼接用的第一張影像到picturebox1(高:300, 寬:400)\*/

private void button1\_Click(object sender, EventArgs e)

{

var dialog = new OpenFileDialog();

dialog.Filter = "影像(\*.jpg/\*.png/\*.gif/\*.bmp)|\*.jpg;\*.png;\*.gif;\*.bmp";

if (dialog.ShowDialog() == DialogResult.OK)

{

var filename = dialog.FileName;

IntPtr image = CvInvoke.cvLoadImage(filename, Emgu.CV.CvEnum.LOAD\_IMAGE\_TYPE.CV\_LOAD\_IMAGE\_ANYCOLOR);

size1 = CvInvoke.cvGetSize(image);

dest1 = new Image<Bgr, byte>(size1);

generate1 = new Image<Bgr, byte>(size1);

CvInvoke.cvCopy(image, dest1, IntPtr.Zero);

pictureBox1.Image = dest1.ToBitmap();

}

}

第二步:

將三張影像合聘到同一張影像，目的是為了方便抓取和記錄每個像素的位置與RGB值，進而幫助我們來找到三張影像的共同點，進行轉移矩陣的運算。

程式碼:

int size4width = pictureBox4.Size.Width;

int size4height = pictureBox4.Size.Height;

/\*用來紀錄經過轉移矩陣，拼接完成後的RGB值\*/

generate4 = new Image<Bgr, byte>(size4width, size4height);

/\*用來紀錄最一開始沒有經過轉移矩陣，單純拼接的每個位置點的像素的RGB值\*/

dest4 = new Image<Bgr, byte>(size4width, size4height);

/\*大小(高:300,寬:1200)，最一開始的單純拼接，用來記錄每個像素的RGB值，RGB值由picturebox1、picturebox2、picturebox3的影像提供\*/

for (int i = 0; i < size4height; i++)

{

for (int j = 0; j < size4width; j++)

{

/\*0-399的RGB值由picturebox1影像提供\*/

if (j >= 0 && j < 400)

{

dest4.Data[i, j, 0] = (byte)(int)dest1.Data[i, j, 0];

dest4.Data[i, j, 1] = (byte)(int)dest1.Data[i, j, 1];

dest4.Data[i, j, 2] = (byte)(int)dest1.Data[i, j, 2];

}

/\*400-799的RGB值由picturebox2影像提供\*/

else if (j >= 400 && j < 800)

{

dest4.Data[i, j, 0] = (byte)(int)dest2.Data[i, j % 400, 0];

dest4.Data[i, j, 1] = (byte)(int)dest2.Data[i, j % 400, 1];

dest4.Data[i, j, 2] = (byte)(int)dest2.Data[i, j % 400, 2];

}

/\*800-1199的RGB值由picturebox3影像提供\*/

else if (j >=800 && j < 1200)

{

dest4.Data[i, j, 0] = (byte)(int)dest3.Data[i, j % 400, 0];

dest4.Data[i, j, 1] = (byte)(int)dest3.Data[i, j % 400, 1];

dest4.Data[i, j, 2] = (byte)(int)dest3.Data[i, j % 400, 2];

}

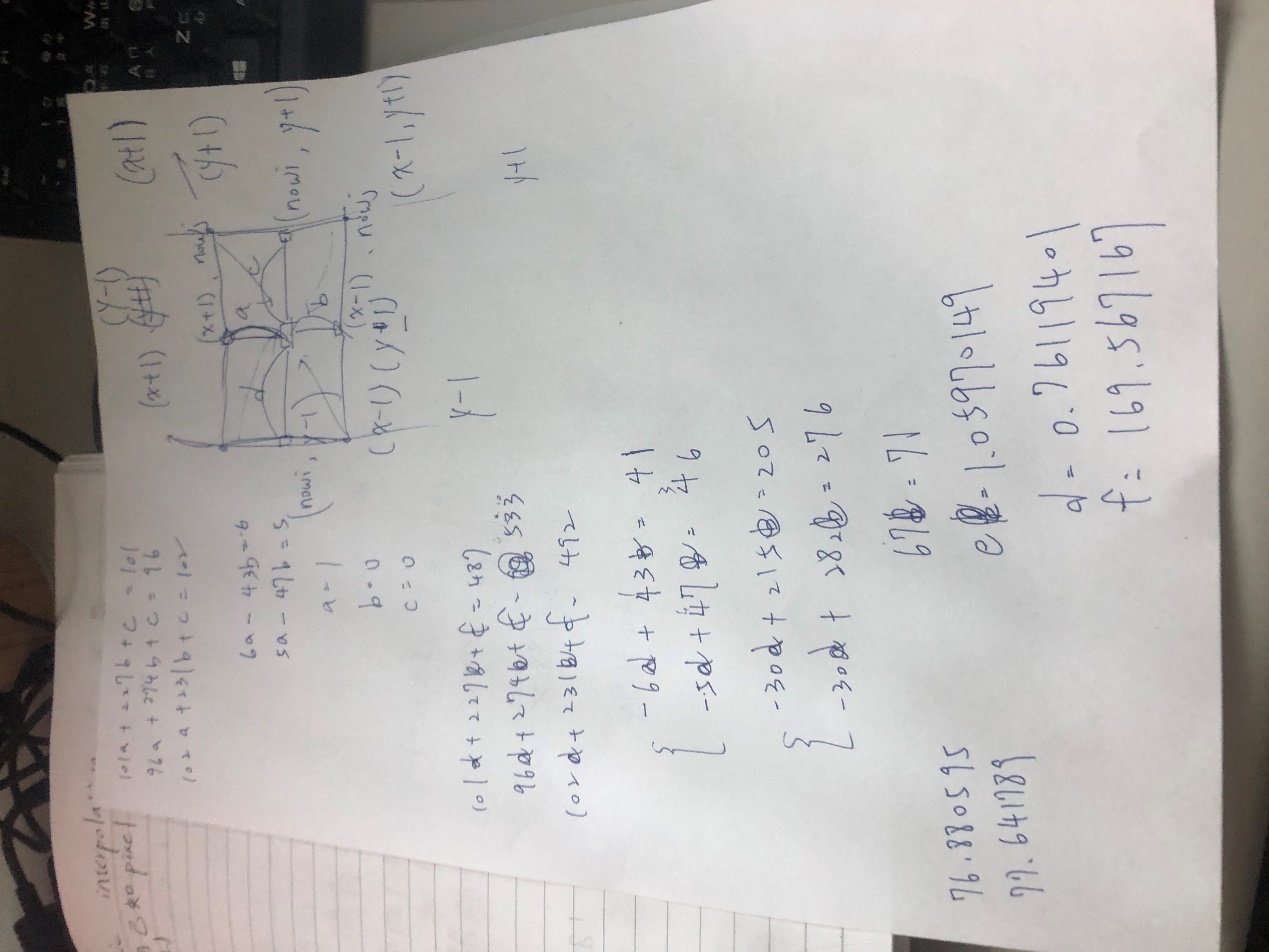
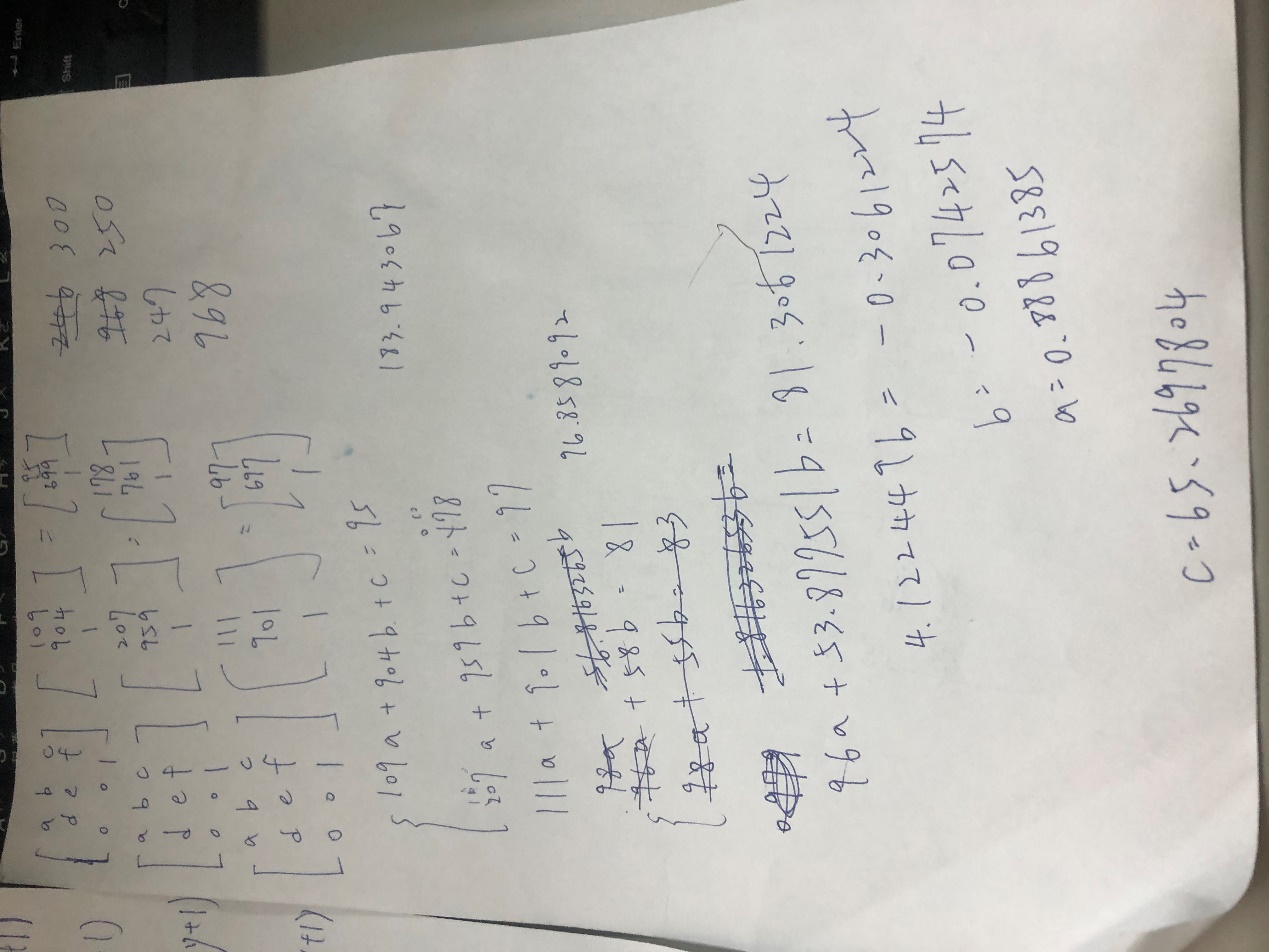
}

/\*將拼接結果顯示在picturebox4\*/

pictureBox4.Image = dest4.ToBitmap();

}

第三步:

算出轉移矩陣，以第二張影像當作基準，接著去尋找第二張影像與第一張影像和第三張影像分別的共同點三個，用小畫家的方式查看像素位置，接著利用矩陣運算反矩陣的方式來求轉移矩陣。

第四步:

求出轉移矩陣後，我們將第一張與第三章的像素位置乘上分別的轉移矩陣，而第二張影像做為基準，所以不用乘上轉移矩陣，所以就像把第一張影像和第三張影像貼到在中間的第二張影像。

程式碼:

函式(乘上轉移矩陣):

/\*將第三張影像的點藉由轉移矩陣轉移到第二張影像\*/

public double[,] moveimage3point(double[,] moveimage3)

{

///double[,] move = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };

double[,] move = new double[3, 3] { { 0.88861385, -0.0742574, 70.2697804 }, { 0.18811882, 0.7920792, -37.5443676 }, { 0, 0, 1 } };

double[,] moveimage3point = new double[3, 1] { { 0 }, { 0 }, { 0 } };

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 1; j++)

{

moveimage3point[i, j] = 0;

for (int k = 0; k < 3; k++)

{

moveimage3point[i, j] += move[i, k] \* moveimage3[k, j];

}

}

}

return moveimage3point;

}

函式(把小數點位置整數化):

/\*由於在將點乘上轉移矩陣，將第一張影像轉移到第二張影像的過程中，點的位置會出現小數點，所以要將點整數化\*/

public int[,] moveimageint(double[,] moveimage2point)

{

int[,] moveint = new int[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 1; j++)

{

moveint[i, j] = Convert.ToInt16(moveimage2point[i, j]);

}

}

return moveint;

}

主程式(每個像素乘上轉移矩陣):

double[,] moveimage1RGB = new double[3, 1];

double[,] moveimage3RGB = new double[3, 1];

double[,] moveimage1pointRGB = new double[3, 1];

double[,] moveimage3pointRGB = new double[3, 1];

int[,] moveimage1intRGB = new int[3, 1];

int[,] moveimage3intRGB = new int[3, 1];

int size5width = pictureBox5.Size.Width;

int size5height = pictureBox5.Size.Height;

/\*對所有的像素的RGB值進行運算，並乘上轉移矩陣\*/

for (int i = 0; i < size5height; i++)

{

for (int j = 0; j < size5width; j++)

{

/\*以第二張影像做中心，直接將RGB值儲存到generate4，並將另外兩張影像貼近第二張影像\*/

if (j > 399 && j < 800)

{

generate4.Data[i, j, 0] = (byte)(int)dest4.Data[i, j, 0];

generate4.Data[i, j, 1] = (byte)(int)dest4.Data[i, j, 1];

generate4.Data[i, j, 2] = (byte)(int)dest4.Data[i, j, 2];

}

/\*對第一張影像的RGB值進行處理，將當下的像素位置乘上轉移矩陣，然後將原本當下的像素RGB值，帶到新的位置\*/

else if (j >= 0 && j < 400)

{

/\*當下處理的像素的位置\*/

moveimage1RGB = new double[3, 1] { { i }, { j }, { 1 } };

/\*將當下像素的位置乘上轉移矩陣，來找到轉移到第二張影像上的哪個位置\*/

moveimage1pointRGB = moveimage1point(moveimage1RGB);

/\*將小數點整數化\*/

moveimage1intRGB = moveimageint(moveimage1pointRGB);

/\*如果當下處理的位置有重疊到第二章影像的部分，不做處理，以第二張影像的為主，其餘的進行轉移處理\*/

if (moveimage1intRGB[0, 0] < 300 && moveimage1intRGB[0, 0] > 0 && moveimage1intRGB[1, 0] < 400 && moveimage1intRGB[1, 0] > 0)

{

generate4.Data[moveimage1intRGB[0, 0], moveimage1intRGB[1, 0], 0] = (byte)(int)dest4.Data[i, j, 0];

generate4.Data[moveimage1intRGB[0, 0], moveimage1intRGB[1, 0], 1] = (byte)(int)dest4.Data[i, j, 1];

generate4.Data[moveimage1intRGB[0, 0], moveimage1intRGB[1, 0], 2] = (byte)(int)dest4.Data[i, j, 2];

}

}

/\*對第三張影像的RGB值進行處理，將當下的像素位置乘上轉移矩陣，然後將原本當下的像素RGB值，帶到新的位置\*/

if (j > 799 && j < 1200)

{

/\*當下處理的像素的位置\*/

moveimage3RGB = new double[3, 1] { { i }, { j }, { 1 } };

/\*將當下像素的位置乘上轉移矩陣，來找到轉移到第二張影像上的哪個位置\*/

moveimage3pointRGB = moveimage3point(moveimage3RGB);

/\*將小數點整數化\*/

moveimage3intRGB = moveimageint(moveimage3pointRGB);

/\*如果當下處理的位置有重疊到第二章影像的部分，不做處理，以第二張影像的為主，其餘的進行轉移處理\*/

if (moveimage3intRGB[0, 0] < 300 && moveimage3intRGB[0, 0] > 0 && moveimage3intRGB[1, 0] >= 800 && moveimage3intRGB[1, 0] < 1200)

{

generate4.Data[moveimage3intRGB[0, 0], moveimage3intRGB[1, 0], 0] = (byte)(int)dest4.Data[i, j, 0];

generate4.Data[moveimage3intRGB[0, 0], moveimage3intRGB[1, 0], 1] = (byte)(int)dest4.Data[i, j, 1];

generate4.Data[moveimage3intRGB[0, 0], moveimage3intRGB[1, 0], 2] = (byte)(int)dest4.Data[i, j, 2];

}

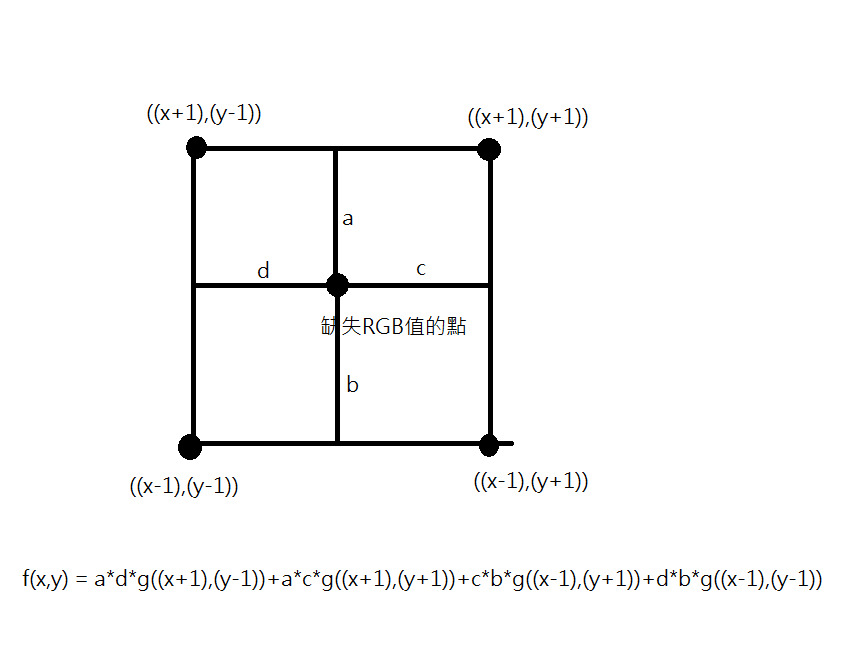
}

}

}

第五步:

內插法，因為在上一步乘上轉移矩陣的過程中，可能會遇到旋轉的問題，導致在轉正為拼接的影像後，有些位置的會沒有RGB值，造成影像會有一點點的空洞，這時我們就可以藉由線性內插的方式，來解決這個問題，首先找到因為轉移後沒有RGB值的點位置，然後找到附近的4個點(下圖)，接著藉由比例分配相乘，來決定這個位置的RGB值。



程式碼:

/\*紀錄當下處理的缺失位置\*/

int nowi = 0, nowj = 0;

/\*紀錄內插需要的4個點\*/

int[] x1 = new int[2] ;

int[] x2 = new int[2] ;

int[] y1 = new int[2] ;

int[] y2 = new int[2] ;

/\*內插四個點的各個比列\*/

double a = 0, b = 0, c = 0, d = 0;

/\*儲存經過內插法後的RGB值\*/

int newr = 0, newg = 0, newb = 0;

/\*內差法，檢查整張照片，有沒有存在沒有存在RGB值的地方\*/

for (int i = 0; i < size5height; i++)

{

for (int j = 0; j < size5width; j++)

{

a = 0;

b = 0;

c = 0;

d = 0;

/\*轉移後的剩下的影像寬度\*/

if (j >= 250 && j <= 920)

{

/\*沒有RGB值的位置\*/

if (generate4.Data[i, j, 0] == 0 && generate4.Data[i, j, 1] == 0 && generate4.Data[i, j, 2] == 0)

{

nowi = i;

nowj = j;

/\*去將當下沒有RGB值的位置往上找到有RGB值存在的地方，並記錄下來\*/

for (int x = nowi; x < 300; x++)

{

if (generate4.Data[x, nowj, 0] != 0 && generate4.Data[x, nowj, 1] != 0 && generate4.Data[x, nowj, 2] != 0)

{

x1[0] = x;

x1[1] = nowj;

break;

}

}

/\*去將當下沒有RGB值的位置往下找到有RGB值存在的地方，並記錄下來\*/

for (int x = nowi; x > 0; x--)

{

if (generate4.Data[x, nowj, 0] != 0 && generate4.Data[x, nowj, 1] != 0 && generate4.Data[x, nowj, 2] != 0)

{

x2[0] = x;

x2[1] = nowj;

break;

}

}

/\*去將當下沒有RGB值的位置往右找到有RGB值存在的地方，並記錄下來\*/

for (int y = nowj; y <= 968; y++)

{

if (generate4.Data[nowi, y, 0] != 0 && generate4.Data[nowi, y, 1] != 0 && generate4.Data[nowi, y, 2] != 0)

{

y1[0] = nowi;

y1[1] = y;

break;

}

}

/\*去將當下沒有RGB值的位置往左找到有RGB值存在的地方，並記錄下來\*/

for (int y = nowj; y >= 250; y--)

{

if (generate4.Data[nowi, y, 0] != 0 && generate4.Data[nowi, y, 1] != 0 && generate4.Data[nowi, y, 2] != 0)

{

y2[0] = nowi;

y2[1] = y;

break;

}

}

int dx = 0;

int dy = 0;

/\*dx為當下的像素往上找到存在RGB值的位置到當下的像素往下找到存在RGB值的位置的距離\*/

dx = x1[0] - x2[0];

/\*dy為當左的像素往上找到存在RGB值的位置到當下的像素往右找到存在RGB值的位置的距離\*/

dy = y1[1] - y2[1];

/\*a為當下像素到往上找到存在RGB值的距離\*/

a = x1[0] - nowi;

/\*如果找不到設為0\*/

if (a < 0)

{

a = 0;

}

/\*如果有找到除以上到下的總距離(dx)，做為內差法的比例\*/

else

{

a /= dx;

}

/\*b為當下像素到往下找到存在RGB值的距離\*/

b = nowi - x2[0];

/\*如果找不到設為0\*/

if (b >= nowi)

{

b = 0;

}

/\*如果有找到除以上到下的總距離(dx)，做為內差法的比例\*/

else

{

b /= dx;

}

/\*c為當下像素到往右找到存在RGB值的距離\*/

c = y1[1] - nowj;

/\*如果找不到設為0\*/

if (c < 0)

{

c = 0;

}

/\*如果有找到除以左到右的總距離(dy)，做為內差法的比例\*/

else

{

c /= dy;

}

/\*d為當下像素到往右找到存在RGB值的距離\*/

d = nowj-y2[1];

/\*如果找不到設為0\*/

if (d >= nowj)

{

d = 0;

}

/\*如果有找到除以左到右的總距離(dy)，做為內差法的比例\*/

else

{

d /= dy;

}

/\*用newr、newg、newb來存放，由4個位置({+1,+1}，{+1,-1}，{-1,-1}，{-1,+1})的RGB值乘上內插法的比列，來決定經過轉移矩陣後，沒有RGB值的像素\*/

newr = (int)(a \* d \* generate4.Data[x1[0], y2[1], 0] + a \* c \* generate4.Data[x1[0], y1[1], 0] + c \* b \* generate4.Data[x2[0], y1[1], 0] + d \* b \* generate4.Data[x2[0], y2[1], 0]);

newg = (int)(a \* d \* generate4.Data[x1[0], y2[1], 1] + a \* c \* generate4.Data[x1[0], y1[1], 1] + c \* b \* generate4.Data[x2[0], y1[1], 1] + d \* b \* generate4.Data[x2[0], y2[1], 1]);

newb = (int)(a \* d \* generate4.Data[x1[0], y2[1], 2] + a \* c \* generate4.Data[x1[0], y1[1], 2] + c \* b \* generate4.Data[x2[0], y1[1], 2] + d \* b \* generate4.Data[x2[0], y2[1], 2]);

generate4.Data[i, j, 0] = (byte)(int)newr;

generate4.Data[i, j, 1] = (byte)(int)newg;

generate4.Data[i, j, 2] = (byte)(int)newb;

}

}

}

}

結果圖:

