به نام خدا

Digital Image Processing Mini Project #2

Geometric transforms of images

Seyed Hesamoddin Hosseini

خلاصه

بناست در این مینی پروژه با استفاده از تبدیلات هندسی مناسب، 4 تصویر رنگی (RGB) با ابعاد 64 x 64 را بر روی چهار بخش ذوزنقه شکل، برابر و خاکستری رنگ یک تصویر بزرگتر و خاکستری (grayscale) با ابعاد 1000 x 1000 قرار دهیم. تبدیلات هندسی را با بکارگیری درونیابی نزدیکترین همسایه و درونیابی دوخطی انجام می دهیم. برای اعمال تبدیل هندسی از **یک مدل دو خطی** استفاده شده است. که در آن باید برای هر یک از چهار تصویر، ابتدا ضرایب 4 معادله 4 مجهول را یک بار برای ردیف ها و یک بار برای ستون ها بدست آورد و سپس به کمک آن ضرایب، نگاشت معکوس را بر روی تصویر هدف اعمال می کنیم. برای پیاده سازی این پروژه از Matlab استفاده می نماییم.

شرح تکنیکال:

مرحله اول: یافتن مختصات corner های ذوزنقه های خاکستری رنگ در تصویر cover.

برای این کار به کمک تعدادی حلقه for و چند شرط ساده، مختصات سطر و ستون گوشه های ذوزنقه ها را بدست می آوریم.

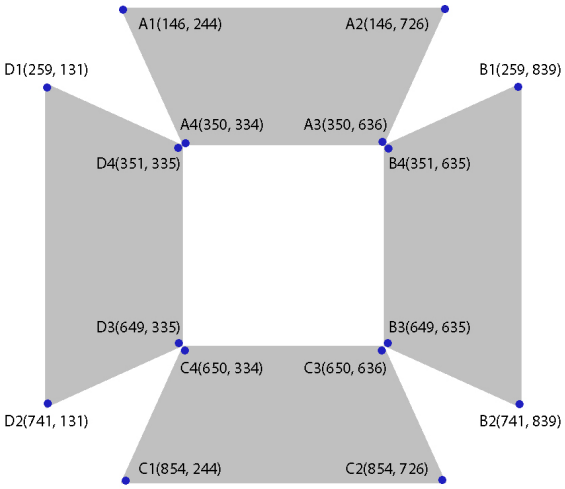

Figure 1 مختصات گوشه های ذوزنقه های تصویر cover

مرحله دوم: پیدا کردن ضرایب دو خطی به کمک حل 4 معادله 4 مجهول

برای ذوزنقه A، به کمک مختصات 4 گوشه آن و مختصات 4 گوشه تصاویر کوچک، 4 معادله برای پیدا کردن ضرایب سطری و 4 معادله برای پیدا کردن ضرایب ستونی به صورت زیر تشکیل می دهیم:

```
eqn1 = c1 * A1_row + c2 * A1_col + c3 * A1_row * A1_col + c4 == 1;
eqn2 = c1 * A2_row + c2 * A2_col + c3 * A2_row * A2_col + c4 == 1;
eqn3 = c1 * A3_row + c2 * A3_col + c3 * A3_row * A3_col + c4 == SmallImage_row_size;
eqn4 = c1 * A4_row + c2 * A4_col + c3 * A4_row * A4_col + c4 == SmallImage_row_size;
```

```
eqn1 = c5 * A1_row + c6 * A1_col + c7 * A1_row * A1_col + c8 == 1;
eqn2 = c5 * A2_row + c6 * A2_col + c7 * A2_row * A2_col + c8 == SmallImage_col_size;
eqn3 = c5 * A3_row + c6 * A3_col + c7 * A3_row * A3_col + c8 == SmallImage_col_size;
eqn4 = c5 * A4_row + c6 * A4_col + c7 * A4_row * A4_col + c8 == 1;
```

مرحله بالا را برای ذوزنقه های B، C و D تکرار می کنیم.

مرحله سوم: نگاشت معکوس

به کمک ضرایب بدست آمده از حل معادلات فوق، نگاشت معکوس را برای تمام پیکسل های روی سطح ذوزنقه به صورت زیر اجرا می نماییم:

```
for row = A1_row : A4_row
    for col = A1_col:A2_col
        if CoverImage(row , col) ~= 255

            x = c1 * row + c2 * col + c3 * row * col + c4;
            y = c5 * row + c6 * col + c7 * row * col + c8;
```

مقدار x و y که در قطعه کد بالا بدست می آید، اگر دقیقا به یک پیکسل از تصویر مبدا اشاره نماید، مقدار سطح خاکستری آن پیکسل از تصویر مبدا را برداشته و در تصویر مقصد جایگذاری می نماییم.

اما اگر x و y بدست آمده، مقایر غیر صحیح بودند. باید درونیابی انجام دهیم.

برای این منظور دو الگوریتم درونیابی نزدیکترین همسایه و درونیابی دو خطی بر روی پیکسل ها اعمال می شود.

برای پیاده سازی الگوریتم درونیابی نزدیک ترین همسایه در متلب، کافیست از تابع round استفاده نماییم. این تابع مقادیر x و y را به عدد صحیحی که به آن نزدیک تر باشد، گرد می نماید.

```
%nearest neighbor
Destination_x = round(x);
Destination_y = round(y);

CoverImage(row , col) = SmallImageA(Destination_x, Destination_y);
```

برای پیاده سازی الگوریتم درون یابی دو خطی، ابتدا باید پارامتر هایی که در شکل 2 نشان داده شده است را محاسبه نماییم.

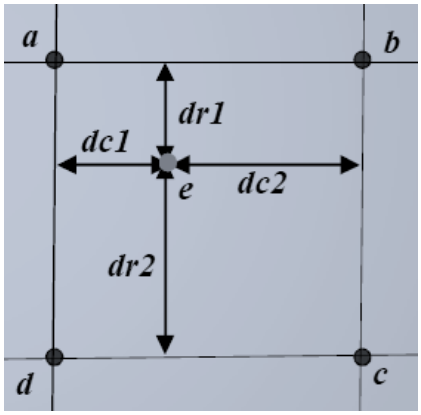

Figure 2 - درونیابی دو خطی

برای این منظور در متلب از تابع floor برای محاسبه کف و از تابع ceil برای محاسبه سقف استفاده می نماییم.

```
a_row = floor(x);
b_row = floor(x);

c_row = ceil(x);
d_row = ceil(x);

a_col = floor(y);
d_col = floor(y);

b_col = ceil(y);
c_col = ceil(y);
```

```
dr1 = x - a_row;
dr2 = d_row - x;
dc1 = y - a_col;
dc2 = b_col - y;
```

پس از بدست آوردن مقادیر فوق، آن ها را در فرمول زیر جایگذاری می نماییم تا مقدار سطح خاکستری e را محاسبه نماییم.

```
e = SmallImageA(a_row, a_col) * dr2 * dc2 + SmallImageA(b_row, b_col) * dr2 * dc1 +
SmallImageA(c_row, c_col) * dr1 * dc1 + SmallImageA(d_row, d_col) * dr1 * dc2;
```

تمامی مراحل فوق را برای هر 4 تصویر تکرار می نماییم.

شرح نتایج:

با اعمال الگوریتم های درونیابی نزدیک ترین همسایه و دو خطی، نتایج متفاوتی بدست می آید که در این بخش به مقایسه آن ها می پردازیم.

الگوریتم درونیابی نزدیک ترین همسایه نسبت به الگوریتم درونیابی دو خطی، پیاده سازی ساده تر و بار محاسباتی کمتری داشت. اما از طرفی باعث شد تصویر نگاشت یافته، به صورت بلاک بلاک و موزاییکی دیده شود و پیکسل ها درشت تر به نظر می رسند.

الگوریتم درونیابی دو خطی نسبت به الگوریتم درونیابی نزدیک ترین همسایه، پیاده سازی پیچیده تر و بار محاسباتی بیشتری داشت. اما از طرفی باعث شد تصویر نگاشت یافته، به صورت بلاک بلاک و موزاییکی دیده نشود. اما شاهد محو شدگی در لبه ها هستیم و لبه ها حالت sharp خود را از دست داده اند.

نتایج:

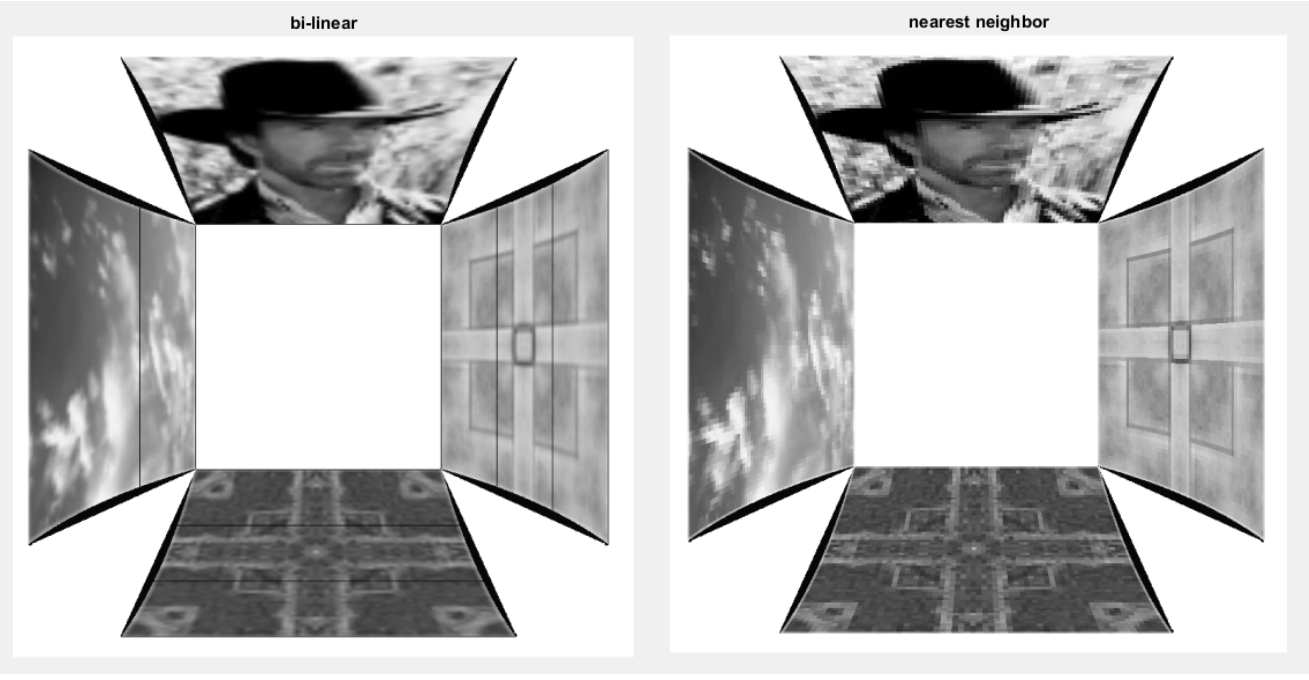

Figure 3 – تصویر سمت راست حاصل نگاشت معکوس و درونیابی نزدیک ترین همسایه – تصویر سمت چپ حاصل نگاشت معکوس و درونیابی دو خطی

پیوست:

سورس کد پروژه با نرم افزار متلب:

```
clear;
```

```
%read images
CoverImage = imread('D:/Im431.bmp');
SmallImageA = imread('D:/Im435.bmp');
SmallImageB = imread('D:/Im434.bmp');
SmallImageC = imread('D:/Im433.bmp');
SmallImageD = imread('D:/Im432.bmp');

SmallImageA = rgb2gray(SmallImageA);
SmallImageB = rgb2gray(SmallImageB);
SmallImageC = rgb2gray(SmallImageC);
SmallImageD = rgb2gray(SmallImageD);

CoverImage2 = CoverImage;

[CoverImage_row_size, CoverImage_col_size] = size(CoverImage);
[SmallImage_row_size, SmallImage_col_size] = size(SmallImageD);

%Determine coordinates of corners

%A

sw = true;

for row = 1:CoverImage_row_size
    for col = 1:CoverImage_col_size

        if CoverImage(row, col) ~= 255
            A1_row = row;
            A1_col = col;

            for col2 = col + 1:CoverImage_col_size
                if CoverImage(row, col2) == 255
                  A2_row = row;
                  A2_col = col2 - 1;
                  break
                end
            end
            sw = false;
            break
        end
    end
    if sw == false
        break
    end
end

%B

sw = true;

for col = CoverImage_col_size:-1:1
    for row = 1:CoverImage_row_size

        if CoverImage(row, col) ~= 255
            B1_row = row;
            B1_col = col;

            for row2 = row + 1:CoverImage_row_size
                if CoverImage(row2, col) == 255
                  B2_row = row2 - 1;
                  B2_col = col;
                  break
                end
```

```
            end
            sw = false;
            break
        end
    end
    if sw == false
        break
    end
end

%C

sw = true;

for row = CoverImage_row_size:-1:1
    for col = 1:CoverImage_col_size

        if CoverImage(row, col) ~= 255
            C1_row = row;
            C1_col = col;

            for col2 = col + 1:CoverImage_col_size
                if CoverImage(row, col2) == 255
                  C2_row = row;
                  C2_col = col2 - 1;
                  break
                end
            end
            sw = false;
            break
        end
    end
    if sw == false
        break
    end
end

%D

sw = true;

for col = 1:CoverImage_col_size
    for row = 1:CoverImage_row_size

        if CoverImage(row, col) ~= 255
            D1_row = row;
            D1_col = col;

            for row2 = row + 1:CoverImage_row_size
                if CoverImage(row2, col) == 255
                  D2_row = row2 - 1;
                  D2_col = col;
                  break
                end
            end
            sw = false;
            break
        end
    end
    if sw == false
        break
    end
end
```

```
%center Square

for row = CoverImage_row_size/2:-1:1
    if CoverImage(row, CoverImage_col_size/2) ~= 255

        A3_row = row;
        A4_row = row;
        B4_row = row + 1;
        D4_row = row + 1;

        break
    end
end

for row = CoverImage_row_size/2:CoverImage_row_size
    if CoverImage(row, CoverImage_col_size/2) ~= 255

        C3_row = row;
        C4_row = row;
        B3_row = row - 1;
        D3_row = row - 1;

        break
    end
end

for col = CoverImage_col_size/2:CoverImage_col_size
    if CoverImage(CoverImage_row_size/2, col) ~= 255

        B3_col = col;
        B4_col = col;
        A3_col = col + 1;
        C3_col = col + 1;

        break
    end
end

for col = CoverImage_col_size/2:-1:1
    if CoverImage(CoverImage_row_size/2, col) ~= 255

        D3_col = col;
        D4_col = col;
        A4_col = col - 1;
        C4_col = col - 1;

        break
    end
end

%Solving the bi-linear 4 equations for image A

syms c1 c2 c3 c4
eqn1 = c1 * A1_row + c2 * A1_col + c3 * A1_row * A1_col + c4 == 1;
eqn2 = c1 * A2_row + c2 * A2_col + c3 * A2_row * A2_col + c4 == 1;
eqn3 = c1 * A3_row + c2 * A3_col + c3 * A3_row * A3_col + c4 == SmallImage_row_size;
eqn4 = c1 * A4_row + c2 * A4_col + c3 * A4_row * A4_col + c4 == SmallImage_row_size;

S = solve([eqn1, eqn2, eqn3, eqn4], [c1, c2, c3, c4]);
c1 = double(S.c1);
c2 = double(S.c2);
c3 = double(S.c3);
c4 = double(S.c4);
```

```
syms c5 c6 c7 c8
eqn1 = c5 * A1_row + c6 * A1_col + c7 * A1_row * A1_col + c8 == 1;
eqn2 = c5 * A2_row + c6 * A2_col + c7 * A2_row * A2_col + c8 == SmallImage_col_size;
eqn3 = c5 * A3_row + c6 * A3_col + c7 * A3_row * A3_col + c8 == SmallImage_col_size;
eqn4 = c5 * A4_row + c6 * A4_col + c7 * A4_row * A4_col + c8 == 1;

S = solve([eqn1, eqn2, eqn3, eqn4], [c5, c6, c7, c8]);

c5 = double(S.c5);
c6 = double(S.c6);
c7 = double(S.c7);
c8 = double(S.c8);

for row = A1_row : A4_row
    for col = A1_col:A2_col

        if CoverImage(row , col) ~= 255

            x = c1 * row + c2 * col + c3 * row * col + c4;
            y = c5 * row + c6 * col + c7 * row * col + c8;

            if isempty(x) == false && isempty(y) == false && x >= 1 && x <=
SmallImage_row_size && y >= 1 && y <= SmallImage_col_size

                %bi-linear

                a_row = floor(x);
                b_row = floor(x);

                c_row = ceil(x);
                d_row = ceil(x);

                a_col = floor(y);
                d_col = floor(y);

                b_col = ceil(y);
                c_col = ceil(y);

                dr1 = x - a_row;
                dr2 = d_row - x;
                dc1 = y - a_col;
                dc2 = b_col - y;

                e = SmallImageA(a_row, a_col) * dr2 * dc2 + SmallImageA(b_row, b_col)
* dr2 * dc1 + SmallImageA(c_row, c_col) * dr1 * dc1 + SmallImageA(d_row, d_col) * dr1
* dc2;

                CoverImage2(row , col) = e;

                %nearest neighbor

                Destination_x = round(x);
                Destination_y = round(y);

                CoverImage(row , col) = SmallImageA(Destination_x, Destination_y);
            end
        end
    end
end

%Solving the bi-linear 4 equations for image B
```

```
syms c1 c2 c3 c4
eqn1 = c1 * B1_row + c2 * B1_col + c3 * B1_row * B1_col + c4 == 1;
eqn2 = c1 * B2_row + c2 * B2_col + c3 * B2_row * B2_col + c4 == 1;
eqn3 = c1 * B3_row + c2 * B3_col + c3 * B3_row * B3_col + c4 == SmallImage_row_size;
eqn4 = c1 * B4_row + c2 * B4_col + c3 * B4_row * B4_col + c4 == SmallImage_row_size;

S = solve([eqn1, eqn2, eqn3, eqn4], [c1, c2, c3, c4]);
c1 = double(S.c1);
c2 = double(S.c2);
c3 = double(S.c3);
c4 = double(S.c4);

syms c5 c6 c7 c8
eqn1 = c5 * B1_row + c6 * B1_col + c7 * B1_row * B1_col + c8 == 1;
eqn2 = c5 * B2_row + c6 * B2_col + c7 * B2_row * B2_col + c8 == SmallImage_col_size;
eqn3 = c5 * B3_row + c6 * B3_col + c7 * B3_row * B3_col + c8 == SmallImage_col_size;
eqn4 = c5 * B4_row + c6 * B4_col + c7 * B4_row * B4_col + c8 == 1;

S = solve([eqn1, eqn2, eqn3, eqn4], [c5, c6, c7, c8]);

c5 = double(S.c5);
c6 = double(S.c6);
c7 = double(S.c7);
c8 = double(S.c8);

for row = B1_row : B2_row
    for col = B4_col:B1_col

        if CoverImage(row , col) ~= 255

            x = c1 * row + c2 * col + c3 * row * col + c4;
            y = c5 * row + c6 * col + c7 * row * col + c8;

            if isempty(x) == false && isempty(y) == false && x >= 1 && x <=
SmallImage_row_size && y >= 1 && y <= SmallImage_col_size

                %bi-linear

                a_row = floor(x);
                b_row = floor(x);

                c_row = ceil(x);
                d_row = ceil(x);

                a_col = floor(y);
                d_col = floor(y);

                b_col = ceil(y);
                c_col = ceil(y);

                dr1 = x - a_row;
                dr2 = d_row - x;
                dc1 = y - a_col;
                dc2 = b_col - y;

                e = SmallImageB(a_row, a_col) * dr2 * dc2 + SmallImageB(b_row, b_col)
* dr2 * dc1 + SmallImageB(c_row, c_col) * dr1 * dc1 + SmallImageB(d_row, d_col) * dr1
* dc2;

                CoverImage2(row , col) = e;

                %nearest neighbor
```

```
                Destination_x = round(x);
                Destination_y = round(y);

                CoverImage(row , col) = SmallImageB(Destination_x, Destination_y);
            end
        end
    end
end

%Solving the bi-linear 4 equations for image A

syms c1 c2 c3 c4
eqn1 = c1 * C1_row + c2 * C1_col + c3 * C1_row * C1_col + c4 == 1;
eqn2 = c1 * C2_row + c2 * C2_col + c3 * C2_row * C2_col + c4 == 1;
eqn3 = c1 * C3_row + c2 * C3_col + c3 * C3_row * C3_col + c4 == SmallImage_row_size;
eqn4 = c1 * C4_row + c2 * C4_col + c3 * C4_row * C4_col + c4 == SmallImage_row_size;

S = solve([eqn1, eqn2, eqn3, eqn4], [c1, c2, c3, c4]);
c1 = double(S.c1);
c2 = double(S.c2);
c3 = double(S.c3);
c4 = double(S.c4);

syms c5 c6 c7 c8
eqn1 = c5 * C1_row + c6 * C1_col + c7 * C1_row * C1_col + c8 == 1;
eqn2 = c5 * C2_row + c6 * C2_col + c7 * C2_row * C2_col + c8 == SmallImage_col_size;
eqn3 = c5 * C3_row + c6 * C3_col + c7 * C3_row * C3_col + c8 == SmallImage_col_size;
eqn4 = c5 * C4_row + c6 * C4_col + c7 * C4_row * C4_col + c8 == 1;

S = solve([eqn1, eqn2, eqn3, eqn4], [c5, c6, c7, c8]);

c5 = double(S.c5);
c6 = double(S.c6);
c7 = double(S.c7);
c8 = double(S.c8);

for row = C4_row : C1_row
    for col = C1_col:C2_col

        if CoverImage(row , col) ~= 255

            x = c1 * row + c2 * col + c3 * row * col + c4;
            y = c5 * row + c6 * col + c7 * row * col + c8;

            if isempty(x) == false && isempty(y) == false && x >= 1 && x <=
SmallImage_row_size && y >= 1 && y <= SmallImage_col_size

                %bi-linear

                a_row = floor(x);
                b_row = floor(x);

                c_row = ceil(x);
                d_row = ceil(x);

                a_col = floor(y);
                d_col = floor(y);

                b_col = ceil(y);
                c_col = ceil(y);

                dr1 = x - a_row;
```

```
                dr2 = d_row - x;
                dc1 = y - a_col;
                dc2 = b_col - y;

                e = SmallImageC(a_row, a_col) * dr2 * dc2 + SmallImageC(b_row, b_col)
* dr2 * dc1 + SmallImageC(c_row, c_col) * dr1 * dc1 + SmallImageC(d_row, d_col) * dr1
* dc2;


                CoverImage2(row , col) = e;

                %nearest neighbor

                Destination_x = round(x);
                Destination_y = round(y);

                CoverImage(row , col) = SmallImageC(Destination_x, Destination_y);
            end
        end
    end
end

%Solving the bi-linear 4 equations for image A

syms c1 c2 c3 c4
eqn1 = c1 * D1_row + c2 * D1_col + c3 * D1_row * D1_col + c4 == 1;
eqn2 = c1 * D2_row + c2 * D2_col + c3 * D2_row * D2_col + c4 == 1;
eqn3 = c1 * D3_row + c2 * D3_col + c3 * D3_row * D3_col + c4 == SmallImage_row_size;
eqn4 = c1 * D4_row + c2 * D4_col + c3 * D4_row * D4_col + c4 == SmallImage_row_size;

S = solve([eqn1, eqn2, eqn3, eqn4], [c1, c2, c3, c4]);
c1 = double(S.c1);
c2 = double(S.c2);
c3 = double(S.c3);
c4 = double(S.c4);

syms c5 c6 c7 c8
eqn1 = c5 * D1_row + c6 * D1_col + c7 * D1_row * D1_col + c8 == 1;
eqn2 = c5 * D2_row + c6 * D2_col + c7 * D2_row * D2_col + c8 == SmallImage_col_size;
eqn3 = c5 * D3_row + c6 * D3_col + c7 * D3_row * D3_col + c8 == SmallImage_col_size;
eqn4 = c5 * D4_row + c6 * D4_col + c7 * D4_row * D4_col + c8 == 1;

S = solve([eqn1, eqn2, eqn3, eqn4], [c5, c6, c7, c8]);

c5 = double(S.c5);
c6 = double(S.c6);
c7 = double(S.c7);
c8 = double(S.c8);

for row = D1_row : D2_row
    for col = D1_col:D4_col

        if CoverImage(row , col) ~= 255

            x = c1 * row + c2 * col + c3 * row * col + c4;
            y = c5 * row + c6 * col + c7 * row * col + c8;

            if isempty(x) == false && isempty(y) == false && x >= 1 && x <=
SmallImage_row_size && y >= 1 && y <= SmallImage_col_size

                %bi-linear

                a_row = floor(x);
                b_row = floor(x);
```

```
                c_row = ceil(x);
                d_row = ceil(x);

                a_col = floor(y);
                d_col = floor(y);

                b_col = ceil(y);
                c_col = ceil(y);

                dr1 = x - a_row;
                dr2 = d_row - x;
                dc1 = y - a_col;
                dc2 = b_col - y;

                e = SmallImageD(a_row, a_col) * dr2 * dc2 + SmallImageD(b_row, b_col)
* dr2 * dc1 + SmallImageD(c_row, c_col) * dr1 * dc1 + SmallImageD(d_row, d_col) * dr1
* dc2;

                CoverImage2(row , col) = e;

                %nearest neighbor

                Destination_x = round(x);
                Destination_y = round(y);

                CoverImage(row , col) = SmallImageD(Destination_x, Destination_y);
            end
        end
    end
end

subplot(1,2,1);
imshow(CoverImage);
title('nearest neighbor');

subplot(1,2,2);
imshow(CoverImage2);
title('bi-linear');
```