原文地址：http://blog.csdn.net/andrew659/article/details/4818988

http://blog.csdn.net/xjz18298268521/article/details/51220576

图像的缩放很好理解,就是图像的放大和缩小。传统的绘画工具中,有一种叫做“放大尺”的绘画工具，画家常用它来放大图画。当然，在计算机上，我们不再需要用放大尺去放大或缩小图像了，把这个工作交给程序来完成就可以了。下面就来讲讲计算机怎么来放大缩小图象；在本文中，我们所说的图像都是指点阵图，也就是用一个像素矩阵来描述图像的方法，对于另一种图像：用函数来描述图像的矢量图，不在本文讨论之列。

越是简单的模型越适合用来举例子，我们就举个简单的图像：3X3 的256级灰度图，也就是高为3个象素，宽也是3个象素的图像，每个象素的取值可以是 0－255，代表该像素的亮度，255代表最亮，也就是白色，0代表最暗，即黑色。假如图像的象素矩阵如下图所示（这个原始图把它叫做源图，Source）：

234　　38　　22

67　　44　　12

89　　65　　63

这个矩阵中，元素坐标(x,y)是这样确定的，x从左到右，从0开始，y从上到下，也是从零开始，这是图象处理中最常用的坐标系，就是这样一个坐标：

----------＞X

|

|

|

|

|

\/Y

如果想把这副图放大为 4X4大小的图像，那么该怎么做呢？那么第一步肯定想到的是先把4X4的矩阵先画出来再说，好了矩阵画出来了，如下所示，当然，矩阵的每个像素都是未知数，等待着我们去填充（这个将要被填充的图的叫做目标图,Destination）：

? ? ? ?

? ? ? ?

? ? ? ?

? ? ? ?

然后要往这个空的矩阵里面填值了，要填的值从哪里来来呢？是从源图中来，好，先填写目标图最左上角的象素，坐标为（0，0），那么该坐标对应源图中的坐标可以由如下公式得
出：

srcX=dstX* (srcWidth/dstWidth) , srcY = dstY * (srcHeight/dstHeight)

好了，套用公式，就可以找到对应的原图的坐标了(0*(3/4),0*(3/4))=>
(0*0.75,0*0.75)=>(0,0)

,找到了源图的对应坐标,就可以把源图中坐标为(0,0)处的234象素值填进去目标图的(0,0)这个位置了。

接下来,如法炮制,寻找目标图中坐标为(1,0)的象素对应源图中的坐标,套用公式:

(1*0.75,0*0.75)=>(0.75,0)

结果发现,得到的坐标里面竟然有小数,这可怎么办?计算机里的图像可是数字图像,象素就是最小单位了,象素的坐标都是整数,从来没有小数坐标。这时候采用的一种策略就是采用四舍五入的方法（也可以采用直接舍掉小数位的方法），把非整数坐标转换成整数，好，那么按照四舍五入的方法就得到坐标（1，0），完整的运算过程就是这样的：

(1*0.75,0*0.75)=>(0.75,0)=>(1,0)

那么就可以再填一个象素到目标矩阵中了，同样是把源图中坐标为(1,0)处的像素值38填入目标图中的坐标。

依次填完每个象素，一幅放大后的图像就诞生了，像素矩阵如下所示：

234	38	22	22
67	44	12	12
89	65	63	63
89	65	63	63

这种放大图像的方法叫做最临近插值算法，这是一种最基本、最简单的图像缩放算法，效果也是最不好的，放大后的图像有很严重的马赛克，缩小后的图像有很严重的失真；效果不好的根源就是其简单的最临近插值方法引入了严重的图像失真，比如，当由目标图的坐标反推得到的源图的的坐标是一个浮点数的时候，采用了四舍五入的方法，直接采用了和这个浮点数最接近的象素的值，这种方法是很不科学的，当推得坐标值为 0.75的时候，不应该就简单的取为1，既然是0.75，比1要小0.25 ，比0要大0.75 ,那么目标象素值其实应该根据这个源图中虚拟的点四周的四个真实的点来按照一定的规律计算出来的，这样才能达到更好的缩放效果。**双线型内插值算法就是一种比较好的图像缩放算法，它充分的利用了源图中虚拟点四周的四个真实存在的像素值来共同决定目标图中的一个像素值，因此缩放效果比简单的最邻近插值要好很多。**

双线性内插值算法描述如下:

对于一个目的像素，设置坐标通过反向变换得到的浮点坐标为(i+u,j+v) (其中i、j均为浮点坐标的整数部分，u、v为浮点坐标的小数部分，是取值[0,1)区间的浮点数)，则这个像素得值 f(i+u,j+v) 可由原图像中坐标为 (i,j)、(i+1,j)、(i,j+1)、(i+1,j+1)所对应的周围四个像素的值决定，即：

f(i+u,j+v) = (1-u)(1-v)f(i,j) + (1-u)vf(i,j+1) + u(1-v)f(i+1,j) + uvf(i+1,j+1) 公式1

其中f(i,j)表示源图像(i,j)处的的像素值，以此类推。

比如，象刚才的例子，现在假如目标图的象素坐标为（1，1），那么反推得到的对应于源图的坐标是（0.75 ，0.75），这其实只是一个概念上的虚拟象素,实际在源图中并不存在这样一个象素,那么目标图的象素（1，1）的取值不能够由这个虚拟象素来决定，而只能由源图的这四个象素共同决定：（0，0）（0，1）（1，0）（1，1），而由于（0.75,0.75）离（1，1）要更近一些，那么（1,1）所起的决定作用更大一些，这从公式1中的系数uv=0.75×0.75就可以体现出来，而（0.75,0.75）离（0，0）最远，所以（0，0）所起的决定作用就要小一些，公式中系数为(1-u)(1-v)=0.25×0.25也体现出了这一特点。

双线性插值

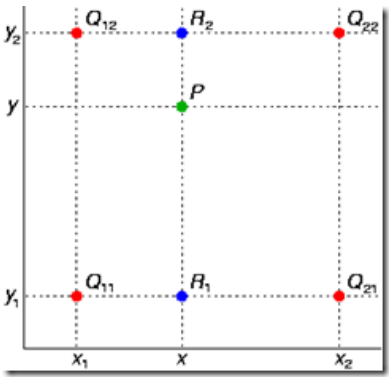
假设源图像大小为mxn，目标图像为axb。那么两幅图像的边长比分别为：m/a和n/b。注意，通常这个比例不是整数，编程存储的时候要用浮点型。目标图像的第（i,j）个像素点（i行j列）可以通过边长比对应回源图像。其对应坐标为（i*m/a,j*n/b）。显然，这个对应坐标一般来说不是整数，而非整数的坐标是无法在图像这种离散数据上使用的。双线性插值通过寻找距离这个对应坐标最近的四个像素点，来计算该点的值（灰度值或者RGB值）。

若图像为灰度图像，那么（i，j）点的灰度值的数学计算模型是：

f(x,y)=b1+b2x+b3y+b4xy

其中b1,b2,b3,b4是相关的系数。关于其的计算过程如下如下：

如图，已知Q12，Q22，Q11，Q21，但是要插值的点为P点，这就要用双线性插值了，首先在x轴方向上，对R1和R2两个点进行插值，这个很简单，然后根据R1和R2对P点进行插值，这就是所谓的双线性插值。



附：维基百科--双线性插值：

双线性插值，又称为双线性内插。在数学上，双线性插值是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。

假如我们想得到未知函数　　在点　　的值，假设我们已知函数　　在　，　，　，及　　四个点的值。

首先在　x　方向进行线性插值，得到

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \quad \text{Where} \quad R_1 = (x, y_1),$$

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \quad \text{Where} \quad R_2 = (x, y_2).$$

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 $f(x, y)$,

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

如果选择一个坐标系统使得　　的四个已知点坐标分别为（0，0）、（0，1）、（1，0）和（1，1），那么插值公式就可以化简为

$$f(x,y) \approx f(0,0)\,(1-x)(1-y) + f(1,0)\,x(1-y) + f(0,1)\,(1-x)y + f(1,1)xy.$$

或者用矩阵运算表示为

$$f(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}$$

这种插值方法的结果通常不是线性的，线性插值的结果与插值的顺序无关。首先进行　y　方向的插值，然后进行　x　方向的插值，所得到的结果是一样的。