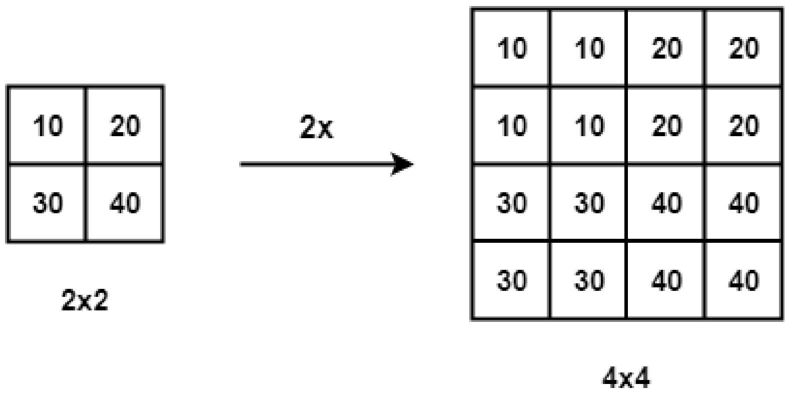
Resizing Images using Various Interpolation Techniques

Interpolation is a method of constructing new data points within the range of a discrete set of known data points. We interpolate (or estimate) the value of that function for an intermediate value of the independent variable.

1. **Nearest Neighbour Interpolation** : To perform interpolation, it is recommended to identify the pixel nearest to the current pixel and determine its value. Subsequently, the values of the known pixels ought to be compared with those of the closest unknown pixels.

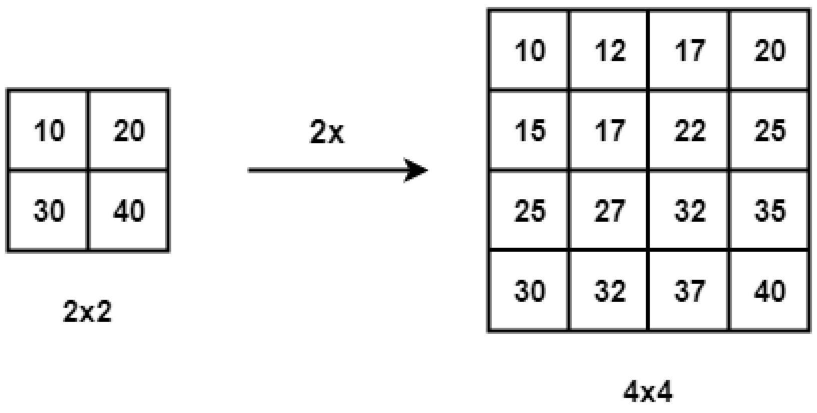


It is observable that the output is a 4x4 image wherein each pixel appears larger in size compared to the original image. This approach involves identifying the nearest neighboring pixel and computing one pixel at a time, resulting in the least processing time.

(Hows it works more:

https://www.youtube.com/watch?v=ORhhP--dNJA)

2. **Bilinear interpolation** : In bilinear interpolation, we take the values of the four nearest known neighbors (2x2 neighborhood) of unknown pixels and then take the average of these values to assign the unknown pixel. Consider the 2x2 image to be projected onto a 4x4 image but only the corner pixels retain the values. The remaining pixels which are technically in the middle of the four are then calculated by using a scale to assign weights depending on the closer pixel.



10	20
30	40

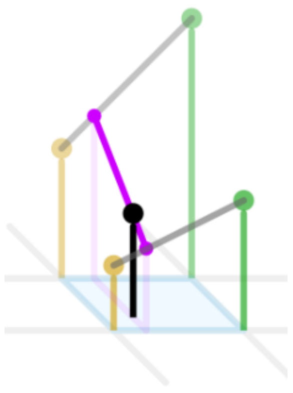
2x2

10	12	17	20
15	17	22	25
25	27	32	35
30	32	37	40

4x4

Bilinear interpolation has a longer processing time than nearest neighbor interpolation as it takes in the value of four pixels to compute the interpolated pixel. However, it gives a smoother output.

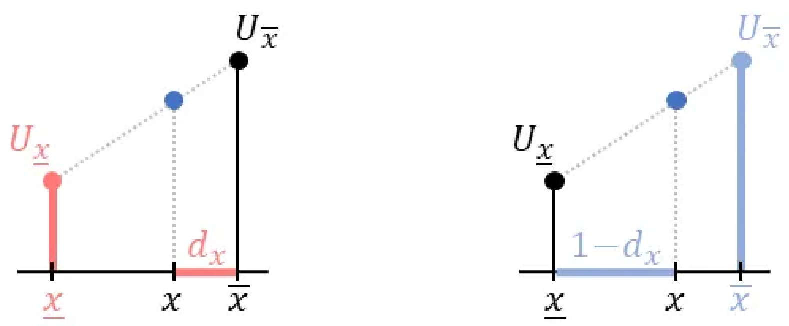
Next, we will discuss the formulation for calculating Bilinear interpolation.



Bilinear

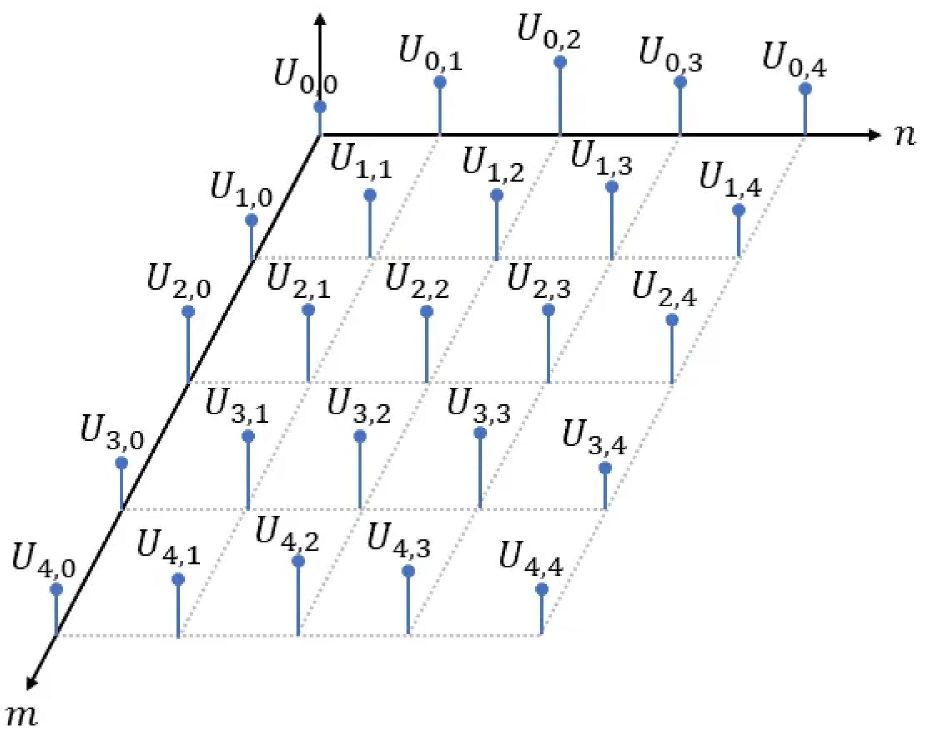
$$U_x = U_{\underline{x}} \cdot d_x + U_{\overline{x}} \cdot (1 - d_x)$$

See how linear interpolation can be interpreted in terms of the values of neighboring points weighted by opposite distances.

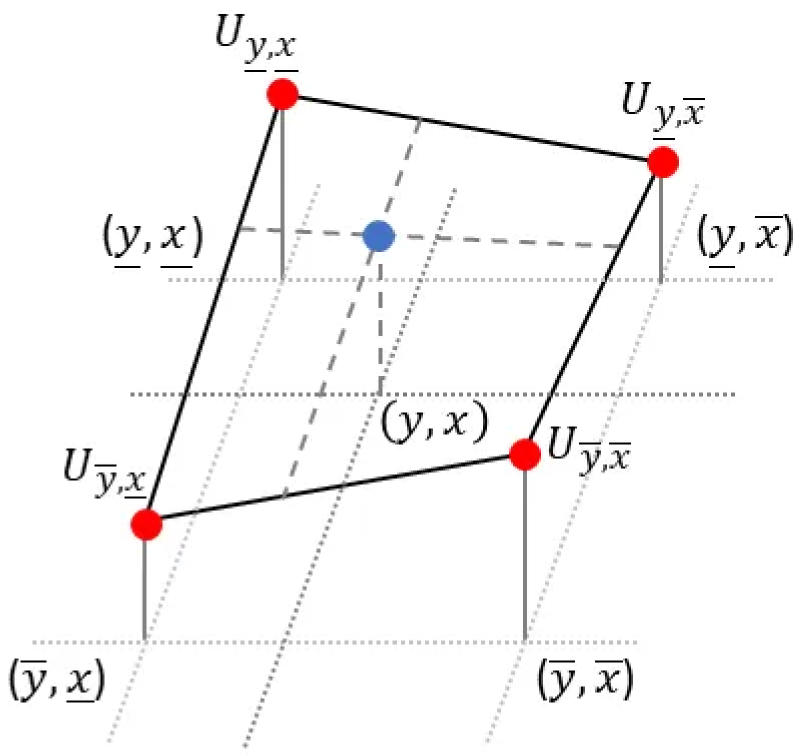


$$U_x = U_{\underline{x}} \cdot d_x + U_{\overline{x}} \cdot (1 - d_x) \qquad U_x = U_{\underline{x}} \cdot d_x + U_{\overline{x}} \cdot (1 - d_x)$$

Next, we will proceed to expand our exploration to the two-dimensional scenario, wherein we are confronted with a collection of data points that are positioned on a uniformly spaced two-dimensional grid.

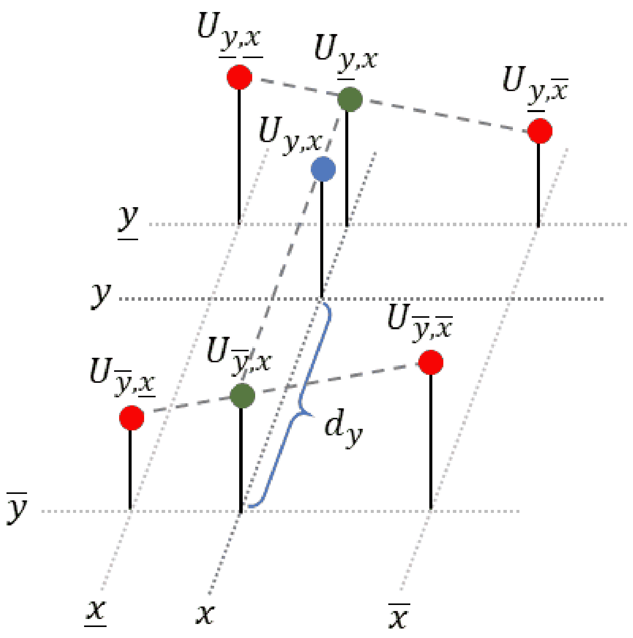


Bilinear interpolation involves fitting a plane to the four nearest neighboring points that encircle the sample point (y, x), followed by a lookup process to obtain the desired value.



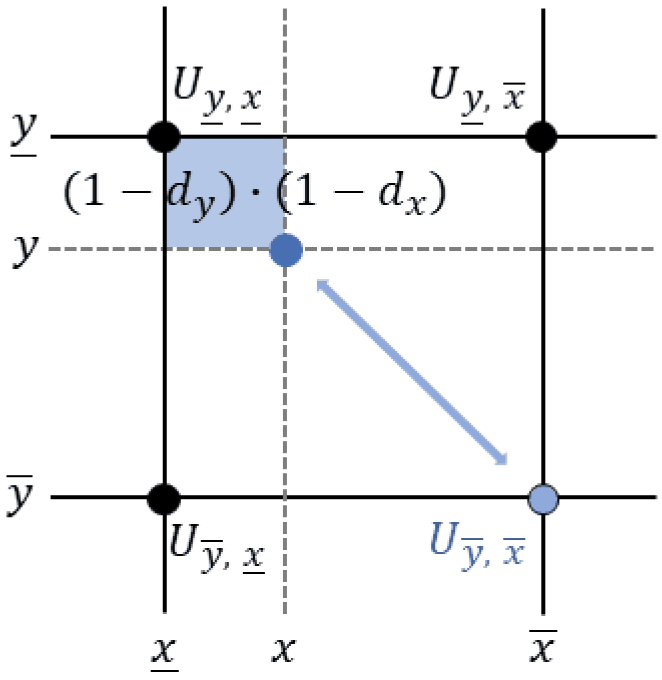
$U_{\underline{y},\underline{x}}$
$U_{\underline{y},\overline{x}}$
$(\underline{y},\underline{x})$
$(\underline{y},\overline{x})$
(y,x)
$U_{\overline{y},\underline{x}}$
$U_{\overline{y},\overline{x}}$
$(\overline{y},\underline{x})$
$(\overline{y},\overline{x})$

Rather than employing bilinear interpolation, we shall iteratively utilize linear interpolation (1-D) to two points in each step. Our procedure begins by performing linear interpolation in the horizontal direction on the upper two points, followed by repeating the process for the lower two points. To obtain the desired value, we perform linear interpolation in the vertical direction on the intermediate points we have just obtained.

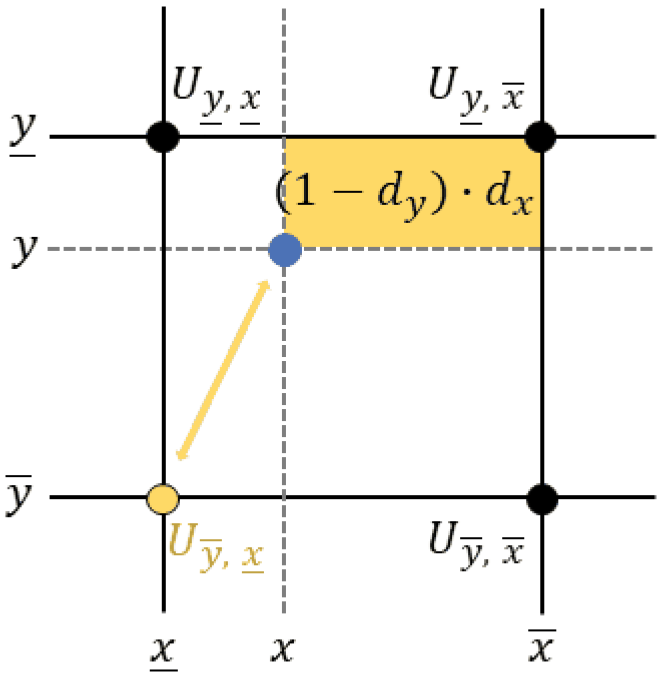


$U_{\underline{y},\underline{x}}$
$U_{\underline{y},x}$
$U_{\underline{y},\overline{x}}$
$U_{y,x}$
$\underline{y}$
y
$U_{\overline{y},\overline{x}}$
$U_{\overline{y},x}$
$U_{\overline{y},\underline{x}}$
d_y
$\overline{y}$
$\underline{x}$
x
$\overline{x}$

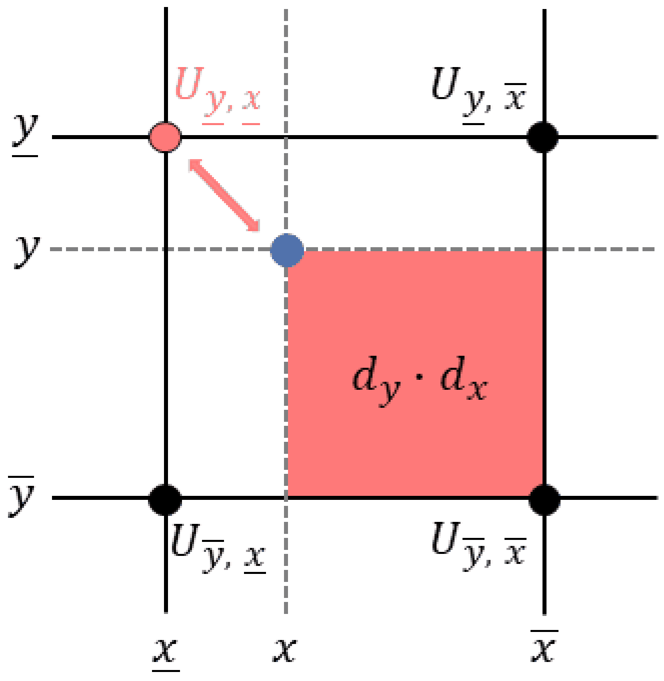
Bilinear interpolation can also be visualized geometrically: to compute the value of the target point (depicted in blue), the sum of the products of the value at each corner and the partial area diagonally opposite the corner needs to be computed.



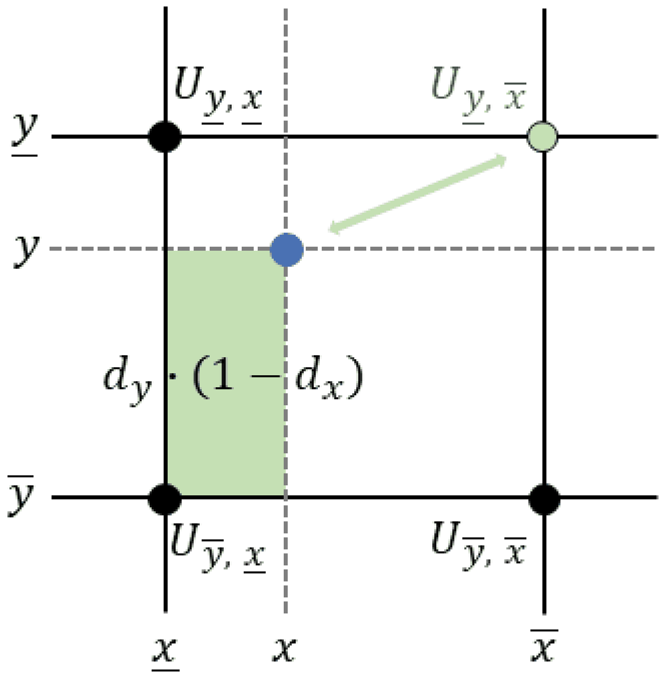
$U_{\underline{y},\underline{x}}$
$U_{\underline{y},\overline{x}}$
$\underline{y}$
$(1-d_y)\cdot(1-d_x)$
y
$\overline{y}$
$U_{\overline{y},\underline{x}}$
$U_{\overline{y},\overline{x}}$
$\underline{x}$
x
$\overline{x}$



$U_{\underline{y},\underline{x}}$
$U_{\underline{y},\overline{x}}$
$\underline{y}$
$(1-d_y)\cdot d_x$
y
$\overline{y}$
$U_{\overline{y},\underline{x}}$
$U_{\overline{y},\overline{x}}$
$\underline{x}$
x
$\overline{x}$



$U_{\underline{y},\underline{x}}$
$U_{\underline{y},\overline{x}}$
$\underline{y}$
y
$d_y \cdot d_x$
$\overline{y}$
$U_{\overline{y},\underline{x}}$
$U_{\overline{y},\overline{x}}$
$\underline{x}$
x
$\overline{x}$



$U_{\underline{y},\underline{x}}$
$U_{\underline{y},\overline{x}}$
$\underline{y}$
y
$d_y \cdot (1 - d_x)$
$\overline{y}$
$U_{\overline{y},\underline{x}}$
$U_{\overline{y},\overline{x}}$
$\underline{x}$
x
$\overline{x}$

There are two other important interpolation techniques called: nearest-neighbor interpolation and bicubic interpolation. The nearest-neighbor interpolation simply takes the nearest pixel to the sample point and copies it to the output location. Bicubic interpolation considers 16 neighbors of the sample point and fits a higher order polynomial to obtain the estimate.

(How it works more: https://www.youtube.com/watch?v=AqscP7rc8_M)

3. **Bicubic Interpolation** : Bicubic interpolation entails considering a 4x4 neighborhood of the pixel to be interpolated, involving a total of 16 pixels. This differs from bilinear interpolation, which only takes into account a 2x2 neighborhood, involving a total of 4 pixels.

   Upon evaluating a 4x4 surface, it is possible to compute the values of the interpolated pixels through the utilization of the following formula :

$$p(x,y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j .$$

The interpolation problem involves determining the 16 coefficients, denoted as ai□. Matching p(x,y) with the function values yields four equations:

1. $f(0,0) = p(0,0) = a_{00},$
2. $f(1,0) = p(1,0) = a_{00} + a_{10} + a_{20} + a_{30},$
3. $f(0,1) = p(0,1) = a_{00} + a_{01} + a_{02} + a_{03},$
4. $f(1,1) = p(1,1) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij}.$

And four equations for the xy mixed partial derivative:

1. $f_{xy}(0,0) = p_{xy}(0,0) = a_{11},$
2. $f_{xy}(1,0) = p_{xy}(1,0) = a_{11} + 2a_{21} + 3a_{31},$
3. $f_{xy}(0,1) = p_{xy}(0,1) = a_{11} + 2a_{12} + 3a_{13},$
4. $f_{xy}(1,1) = p_{xy}(1,1) = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} ij.$

Grouping the unknown parameters a_ij in a vector

$$\alpha = \begin{bmatrix} a_{00} & a_{10} & a_{20} & a_{30} & a_{01} & a_{11} & a_{21} & a_{31} & a_{02} & a_{12} & a_{22} & a_{32} & a_{03} & a_{13} & a_{23} & a_{33} \end{bmatrix}^T$$

and letting

$$x = \begin{bmatrix} f(0,0) & f(1,0) & f(0,1) & f(1,1) & f_x(0,0) & f_x(1,0) & f_x(0,1) & f_x(1,1) & f_y(0,0) & f_y(1,0) & f_y(0,1) & f_y(1,1) & f_{xy}(0,0) & f_{xy}(1,0) & f_{xy}(0,1) & f_{xy}(1,1) \end{bmatrix}^T,$$

The above system of equations can be reformulated into a matrix for the linear equation Aa= x. Inverting the matrix gives the more useful linear equation A^-1x = a, where

$$A^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 \\ -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 \\ 9 & -9 & -9 & 9 & 6 & 3 & -6 & -3 & 6 & -6 & 3 & -3 & 4 & 2 & 2 & 1 \\ -6 & 6 & 6 & -6 & -3 & -3 & 3 & 3 & -4 & 4 & -2 & 2 & -2 & -2 & -1 & -1 \\ 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ -6 & 6 & 6 & -6 & -4 & -2 & 4 & 2 & -3 & 3 & -3 & 3 & -2 & -1 & -2 & -1 \\ 4 & -4 & -4 & 4 & 2 & 2 & -2 & -2 & 2 & -2 & 2 & -2 & 1 & 1 & 1 & 1 \end{bmatrix},$$

Which allows a to be calculated quickly and easily.

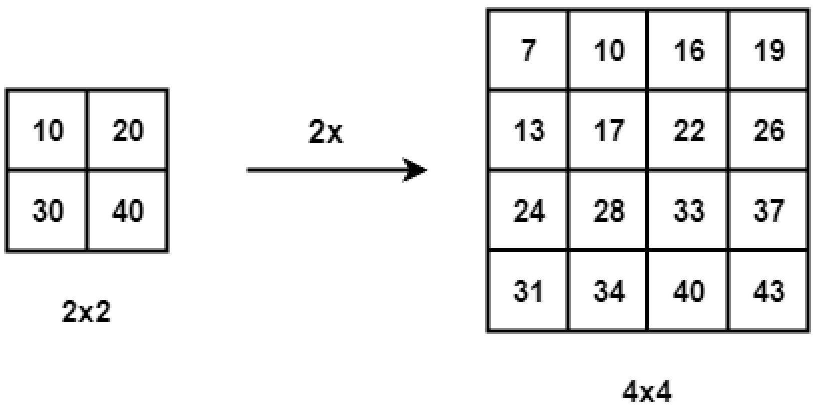
There can be another concise matrix form for 16 coefficients:

$$\begin{bmatrix} f(0,0) & f(0,1) & f_y(0,0) & f_y(0,1) \\ f(1,0) & f(1,1) & f_y(1,0) & f_y(1,1) \\ f_x(0,0) & f_x(0,1) & f_{xy}(0,0) & f_{xy}(0,1) \\ f_x(1,0) & f_x(1,1) & f_{xy}(1,0) & f_{xy}(1,1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 3 \end{bmatrix},$$

where

$$p(x,y) = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 \\ y \\ y^2 \\ y^3 \end{bmatrix}.$$

These coefficients can be computed by utilizing the p(x,y) values derived from the matrix of pixels and the partial derivatives of individual pixels. Subsequently, upon calculating the coefficients, they are multiplied by the weights of the known pixels to interpolate the unknown pixels. Employing bicubic interpolation results in the following output :



10	20
30	40

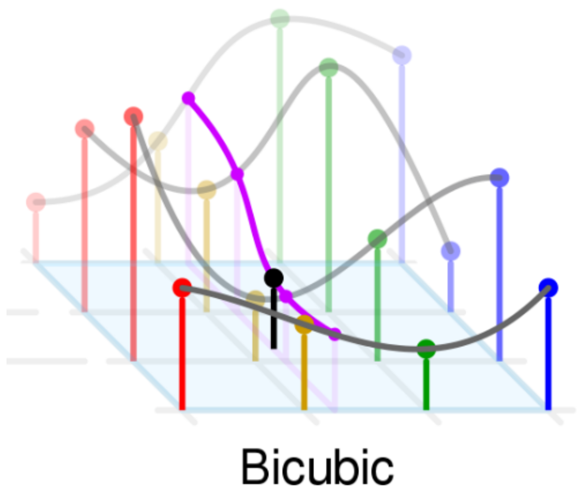
2x2

7	10	16	19
13	17	22	26
24	28	33	37
31	34	40	43

4x4

The utilization of bicubic interpolation produces images that exhibit considerably sharper quality as compared to the previous two interpolation methods, while still maintaining a balance between processing time and output quality. This approach is commonly adopted in various editing software programs, printer drivers, and in-camera interpolation techniques.

Next, we will discuss the formulation for calculating Bilinear interpolation.



Writing the Interpolation Kernel Function for Bicubic Interpolation: The interpolation kernel for bicubic is of the form:

$$W(x) = \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1 & \text{for } |x| \leq 1, \\ a|x|^3 - 5a|x|^2 + 8a|x| - 4a & \text{for } 1 < |x| < 2, \\ 0 & \text{otherwise,} \end{cases}$$

Where a is usually set to -0.5 or -0.75. Note that W(0) and W(n) = 0 for all nonzero integers n.

Writing the bicubic interpolation function: Use the values of 16 pixels around the new pixel dst(x,y). (x,y) shows the location of pixels.

$$dst(x,y) = \begin{pmatrix} u(x_1) & u(x_2) & u(x_3) & u(x_4) \end{pmatrix} \begin{pmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{pmatrix} \begin{pmatrix} u(y_1) \\ u(y_2) \\ u(y_3) \\ u(y_4) \end{pmatrix}$$

Here, f means the values of pixels. x1, x2, x3, x4 are the distance of x direction from new pixel to near 16 pixels. y1, ... are the distance of y direction.

If we use the matrix notation for the common case a = -0.5, we can express the equation in a more friendly manner:

$$p(t) = \tfrac{1}{2}\begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} f_{-1} \\ f_0 \\ f_1 \\ f_2 \end{bmatrix}$$

for t between 0 and 1 for one dimension. Note that for 1-dimensional cubic convolution interpolation 4 sample points are required. For each inquiry two samples are located on its left and two samples on the right. These points are indexed from −1 to 2 in this text. The distance from the point indexed with 0 to the inquiry point is denoted by t here.

For two dimensions first applied once in x and again in y:

$$\begin{aligned} b_{-1} &= p(t_x, f_{(-1,-1)}, f_{(0,-1)}, f_{(1,-1)}, f_{(2,-1)}), \\ b_0 &= p(t_x, f_{(-1,0)}, f_{(0,0)}, f_{(1,0)}, f_{(2,0)}), \\ b_1 &= p(t_x, f_{(-1,1)}, f_{(0,1)}, f_{(1,1)}, f_{(2,1)}), \\ b_2 &= p(t_x, f_{(-1,2)}, f_{(0,2)}, f_{(1,2)}, f_{(2,2)}), \\ p(x,y) &= p(t_y, b_{-1}, b_0, b_1, b_2). \end{aligned}$$

(How it works more: https://www.youtube.com/watch?v=poY_nGzEEWM)

Reference :

https://www.simplilearn.com/tutorials/statistics-tutorial/bilinear-interpolation (What is Bilinear Interpolation?)

https://annmay10.medium.com/resizing-images-using-various-interpolation-techniques-4b99800999f2 (Resizing Images using Various Interpolation Techniques)

https://towardsdatascience.com/spatial-transformer-networks-tutorial-part-2-bilinear-interpolation-371e1d5f164f (Spatial Transformer Networks Tutorial, Part 2 — Bilinear Interpolation)

https://theailearner.com/tag/nearest-neighbor-interpolation/ (Image Demosaicing or Interpolation methods)