Bilinear interpolation

From Wikipedia, the free encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

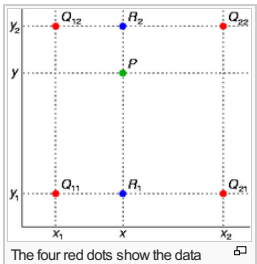
Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Čeština
Español
فارسی
Français
Polski
Русский
Suomi
Українська
中文
Edit links

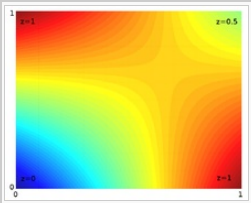
In mathematics, **bilinear interpolation** is an extension of linear interpolation for interpolating functions of two variables (e.g., *x* and *y*) on a rectilinear 2D grid.

The key idea is to perform linear interpolation first in one direction, and then again in the other direction. Although each step is linear in the sampled values and in the position, the interpolation as a whole is not linear but rather quadratic in the sample location.



The four red dots show the data points and the green dot is the point at which we want to interpolate.

Algorithm [edit]



Example of bilinear interpolation on the unit square with the z-values 0, 1, 1 and 0.5 as indicated. Interpolated values in between represented by color.

Suppose that we want to find the value of the unknown function *f* at the point (*x*, *y*). It is assumed that we know the value of *f* at the four points $Q_{11} = (x_1, y_1)$, $Q_{12} = (x_1, y_2)$, $Q_{21} = (x_2, y_1)$, and $Q_{22} = (x_2, y_2)$.

We first do linear interpolation in the *x*-direction. This yields

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

We proceed by interpolating in the *y*-direction to obtain the desired estimate:

$$\begin{aligned} f(x, y) &\approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2) \\ &\approx \frac{y_2 - y}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \right) + \frac{y - y_1}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \right) \\ &= \frac{1}{(x_2 - x_1)(y_2 - y_1)} \left(f(Q_{11})(x_2 - x)(y_2 - y) + f(Q_{21})(x - x_1)(y_2 - y) + f(Q_{12})(x_2 - x)(y - y_1) + f(Q_{22})(x - x_1)(y - y_1) \right) \end{aligned}$$

Note that we will arrive at the same result if the interpolation is done first along the *y*-direction and then along the *x*-direction.

Alternative algorithm [edit]

An alternative way to write the solution to the interpolation problem is

$$f(x, y) \approx a_0 + a_1 x + a_2 y + a_3 xy$$

Where the coefficients are found by solving the linear system

$$\begin{bmatrix} 1 & x_1 & y_1 & x_1 y_1 \\ 1 & x_1 & y_2 & x_1 y_2 \\ 1 & x_2 & y_1 & x_2 y_1 \\ 1 & x_2 & y_2 & x_2 y_2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} f(Q_{11}) \\ f(Q_{12}) \\ f(Q_{21}) \\ f(Q_{22}) \end{bmatrix}$$

If a solution is preferred in terms of *f(Q)* then we can write

$$f(x, y) \approx b_{11} f(Q_{11}) + b_{12} f(Q_{12}) + b_{21} f(Q_{21}) + b_{22} f(Q_{22})$$

Where the coefficients are found by solving

$$\begin{bmatrix} b_{11} \\ b_{12} \\ b_{21} \\ b_{22} \end{bmatrix} = \left(\begin{bmatrix} 1 & x_1 & y_1 & x_1 y_1 \\ 1 & x_1 & y_2 & x_1 y_2 \\ 1 & x_2 & y_1 & x_2 y_1 \\ 1 & x_2 & y_2 & x_2 y_2 \end{bmatrix}^{-1} \right)^T \begin{bmatrix} 1 \\ x \\ y \\ xy \end{bmatrix}$$

Unit Square [edit]

If we choose a coordinate system in which the four points where *f* is known are $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$, then the interpolation formula simplifies to

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

Or equivalently, in matrix operations:

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

Nonlinear [edit]

Contrary to what the name suggests, the bilinear interpolant is *not* linear; but it is the product of two linear functions.

Alternatively, the interpolant can be written as

$$f(x, y) = \sum_{i=0}^{1} \sum_{j=0}^{1} a_{ij} x^i y^j = a_{00} + a_{10} x + a_{01} y + a_{11} xy$$

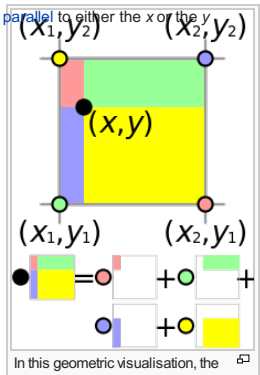
where

$$\begin{aligned} a_{00} &= f(0,0) \\ a_{10} &= f(1,0) - f(0,0) \\ a_{01} &= f(0,1) - f(0,0) \\ a_{11} &= f(1,1) + f(0,0) - (f(1,0) + f(0,1)) \end{aligned}$$

In both cases, the number of constants (four) correspond to the number of data points where *f* is given. The interpolant is linear along lines parallel to either the *x* or the *y* direction, equivalently if *x* or *y* is set constant. Along any other straight line, the interpolant is quadratic. However, even if the interpolation is *not* linear in the position (*x* and *y*), it *is* linear in the amplitude, as it is apparent from the equations above: all the coefficient *bj, j=1..4*, are proportional to the value of the function *f(,)*.

The result of bilinear interpolation is independent of which axis is interpolated first and which second. If we had first performed the linear interpolation in the *y*-direction and then in the *x*-direction, the resulting approximation would be the same.

The obvious extension of bilinear interpolation to three dimensions is called trilinear interpolation.



In this geometric visualisation, the value at the black spot is the sum of the value at each coloured spot multiplied by the area of the rectangle of the same colour, divided by the total area of all four rectangles.

Application in image processing [edit]

In computer vision and image processing, bilinear interpolation is one of the basic resampling techniques.

In texture mapping, it is also known as bilinear filtering or *bilinear texture mapping*, and it can be used to produce a reasonably realistic image. An algorithm is used to map a screen pixel location to a corresponding point on the texture map. A weighted average of the attributes (color, alpha, etc.) of the four surrounding texels is computed and applied to the screen pixel. This process is repeated for each pixel forming the object being textured.[1]

When an image needs to be scaled up, each pixel of the original image needs to be moved in a certain direction based on the scale constant. However, when scaling up an image by a non-integral scale factor, there are pixels (i.e., *holes*) that are not assigned appropriate pixel values. In this case, those *holes* should be assigned appropriate RGB or grayscale values so that the output image does not have non-valued pixels.

Bilinear interpolation can be used where perfect image transformation with pixel matching is impossible, so that one can calculate and assign appropriate intensity values to pixels. Unlike other interpolation techniques such as nearest neighbor interpolation and bicubic interpolation, bilinear interpolation uses only the 4 nearest pixel values which are located in diagonal directions from a given pixel in order to find the appropriate color intensity values of that pixel.

Bilinear interpolation considers the closest 2x2 neighborhood of known pixel values surrounding the unknown pixel's computed location. It then takes a weighted average of these 4 pixels to arrive at its final, interpolated value. The weight on each of the 4 pixel values is based on the computed pixel's distance (in 2D space) from each of the known points.[2]

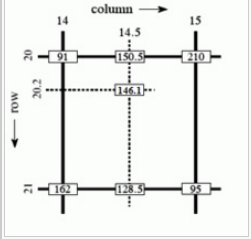
As seen in the example on the right, the intensity value at the pixel computed to be at row 20.2, column 14.5 can be calculated by first linearly interpolating between the values at column 14 and 15 on each rows 20 and 21, giving

$$I_{20,14.5} = \frac{15-14.5}{15-14} \cdot 91 + \frac{14.5-14}{15-14} \cdot 210 = 150.5$$
$$I_{21,14.5} = \frac{15-14.5}{15-14} \cdot 162 + \frac{14.5-14}{15-14} \cdot 95 = 128.5$$

and then interpolating linearly between these values, giving

$$I_{20.2,14.5} = \frac{21-20.2}{21-20} \cdot 150.5 + \frac{20.2-20}{21-20} \cdot 128.5 = 146.1$$

This algorithm reduces some of the visual distortion caused by resizing an image to a non-integral zoom factor, as opposed to nearest neighbor interpolation, which will make some pixels appear larger than others in the resized image. Bilinear interpolation tends, however, to produce a greater number of interpolation artifacts (such as aliasing, blurring, and edge halos) than more computationally demanding techniques such as bicubic interpolation.[3]



Example of bilinear interpolation in grayscale values.

See also [edit]

- Bicubic interpolation
- Trilinear interpolation
- Spline interpolation
- Lanczos resampling
- Stairstep interpolation
- Barycentric coordinates - for interpolating within a triangle or tetrahedron

References [edit]

1. ^ Bilinear interpolation definition at www.pcmag.com
2. ^ "Digital Image Interpolation"
3. ^ "Understanding image-interpolation techniques"

Categories: Multivariate interpolation