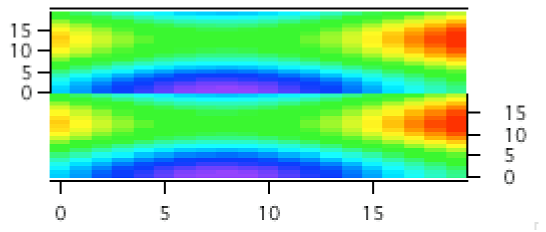
Now we peek at some of the image information for the second image plot of the wave "jack" (which has an instance number of 1) displayed in the top graph:

```
Print ImageInfo("","jack",1)[69,148]        // Just the interesting stuff
```

```
;ZWAVE:jack;ZWAVEDF:root:;COLORMODE:1;RECREATION:ctab= {*,*,Rainbow,0};plane= 0;
```

```
// Apply the color table, etc from jack#1 to jack:
String info= WMGetRECREATIONFromInfo(ImageInfo("","jack",1))
info= RemoveEnding(info)                    // Remove trailing semicolon
```

```
// Use comma instead of semicolon separators
String text = ReplaceString(";", info, ",")
Execute "ModifyImage jack " + text
```



Example 2

This example gets the full path to the wave containing the Z data from which the first image plot in the top graph was calculated.

```
String info= ImageInfo("","",0)      // 0 is index of first image plot
String pathToZ= StringByKey("ZWAVEDF",info)+StringByKey("ZWAVE",info)
Print pathToZ
   root:jack
```

See Also

The **ModifyImage**, **AppendImage**, **NewImage** and **Execute** operations.

Image Plots on page II-385.

Image Instance Names on page II-403.

ImageInterpolate

`ImageInterpolate` `[`***`flags`***`]` ***`Method srcWave`***

The ImageInterpolate operation interpolates the source *srcWave* and stores the results in the wave M_InterpolatedImage in the current data folder unless you specify a different destination wave using the /DEST flag.

Parameters

Method selects type of interpolation. *Method* is one of the following names:

Affine2D	Performs an affine transformation on *srcWave* using parameters specified by the /APRM flag. The transformation applies to a general combination of rotation, scaling, and translation represented by a 3x3 matrix $$M = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & w \end{bmatrix}.$$ The upper 2x2 matrix is a composite of rotation and scaling, *tx* and *ty* are composite translations and *w* is usually 1. It computes the dimensions of the output wave and then uses the inverse transformation and bilinear interpolation to compute the value of each output pixel. When an output pixel does not map back into the source domain it is set to the user-specified background value. It supports 2D and 3D input waves. If *srcWave* is a 3D wave it applies on a layer by layer basis. The output is stored in the wave M_Affine in the current data folder.
Bilinear	Performs a bilinear interpolation subject to the specified flag. You can use either the /F or /S flag, but not both.
Kriging	Uses Kriging to generate an interpolated matrix from a sparse data set. Kriging calculates interpolated values for a rectangular domain specified by the /S flag. The Kriging parameters are specified via the /K flag. Kriging is computed globally for a single user-selected variogram model. If there are significant spatial variances within the domain occupied by the data, you should consider subdividing the domain along natural boundaries and use a single variogram model in each subdivision. If there are N data points, the algorithm first computes the NxN matrix containing the distances between the data and then inverts an associated matrix of similar size to compute the result for the selected variogram model. Because inversion of an NxN matrix can be computationally expensive, you should consider restricting the calculation to regions that are similar to the range implied by the variogram. Such an approach can also be justified in the sense that the local interpolation should not be affected by a remote datum. **Note**: Kriging does not support data containing NaNs or INFs. Wave scaling has no effect.
Pixelate	Creates a lower resolution (pixelated) version of *srcWave* by averaging the pixels inside domain specified by /PXSZ flag. The results are saved in the wave M_PixelatedImage in the current data folder. The computed wave has the same numeric type as *srcWave* so the averaging may be inaccurate in the case of integer waves. When *srcWave* is a 3D wave you have the option of averaging only over data in each layer using /PXSZ={nx,ny} or averaging over data in a rectangular cube using /PXSZ={nx,ny,nz}. When not averaging over layers, the number of rows and columns of the new image are obtained by integer division of the original number by the respective size of the averaging rectangle and adding one more pixel for any remainder. When averaging over layers the resulting rows and columns are obtained by truncated integer division ignoring remainders (if any). See also **Multidimensional Decimation** on page II-98.

Resample	Computes a new image based on the selected interpolation function and transformation parameters. Set the interpolation function with the /FUNC flag. Use the /TRNS flag to specify transformation parameters for grayscale images, or /TRNR, /TRNG, and /TRNB for the red, green, and blue components, respectively, of RGB images. M_InterpolatedImage contains the output image in the current data folder. There are currently two transformation functions: the first magnifies an image and the second applies a radial polynomial sampling. The radial polynomial affects pixels based on their position relative to the image center. A linear polynomial reproduces the same image. Any nonlinear terms contribute to distortion (or correction thereof).
Spline	Computes a 2D spline interpolation for 2D matrix data. The degree of the spline is specified by the /D flag.
Voronoi	Generates an interpolated matrix from a sparse data set (*srcWave* must be a triplet wave) using Voronoi polygons. It calculates interpolated values for a rectangular domain as specified by the /S or /RESL flags. It first computes the Delaunay triangulation of X, Y locations in the Z=0 plane (assuming that X, Y positions occupy a convex domain in the plane). It then uses the Voronoi dual to interpolate the Z values for X and Y pairs from the grid defined by the /S flag. The computed grid may exceed the bounds of the convex domain defined by the triangulation. Interpolated values for points outside the convex domain are set to NaN or the value specified by the /E flag. Use the /I flag to iterate to finer triangulation by subdividing the original triangles into smaller domains. Each iteration increases computation time by approximately a factor of two, but improves the smoothness of the interpolation. If you have multiple sets of data in which X,Y locations are unchanged, you can use the /STW flag to store one triangulation and then use the /PTW flag to apply the precomputed triangulation to a new interpolation. To use this option you should use the Voronoi keyword first with a triplet wave for *srcWave* and set xn = x0 and yn = y0. The operation creates the wave W_TriangulationData that you use in the next triangulation with a 1D wave as *srcWave*. Voronoi interpolation is similar to what can be accomplished with the **ContourZ** function except that it does not require an existing contour plot, it computes the whole output matrix in one call, and it has the option of controlling the subdivision iterations. See **Voronoi Interpolation Example** below.

XYWaves	Performs bilinear interpolation on a matrix scaled using two X and Y 1D waves (specified by /W). The interpolation range is defined by /S. The data domain is defined between the centers of the first and last pixels (X in this example):
	`xmin=(xWave[0]+xWave[1])/2`
	`xmax=(xWave[last]+xWave[last-1])/2`
	Values outside the domain of the data are set to NaN. The interpolation is contained in the M_InterpolatedImage wave, which is single precision floating point or double precision if *srcWave* is double precision.

Warp	Performs image warping interpolation using a two step algorithm with three optional interpolation methods. The operation warps the image based on the relative positions of source and destination grids. The warped image has the same size as the source image. The source and destination grids are each specified by a pair of 2D X and Y waves where the rows and columns correspond to the relative location of the source grid. The smallest supported dimensions of grid waves are 2x2. All grid waves must be double-precision floating point and must have the same number of points corresponding to pixel positions within the image. Grid waves must not contain NaNs or INFs. Wave scaling is ignored.

Flags

/APRM={*r11*,*r12*,*tx*,*r21*,*r22*,*ty*,*w*,*background*}

Sets elements of the affine transformation matrix and the background value.

/ATOL	Allows ImageInterpolate to use a tolerance value if the result of the interpolation at any point is NaN. The algorithm returns the first non-NaN value it finds by adding or subtracting 1/10000th of the size of the X or Y interpolation step. At worst this algorithm is 5 times slower than the default algorithm if the interpolation is performed in a region which is completely outside the convex source domain. /ATOL was added in Igor Pro 7.00.
/CMSH	Use this flag in Voronoi interpolation to create a triangle mesh surface representing the input data. After triangulating the X, Y locations (in a plane z=const), the mesh is generated from a sequence of XYZ vertices of all the resulting triangles. The output is stored in the wave M_ScatterMesh in the current data folder. It is in the form of a triplet wave where every consecutive 3 rows represent a disjoint triangle. If the input contains a NaN or INF in any column, the corresponding row is excluded from the triangulation. If you want to generate this mesh without generating the interpolated matrix you can omit the /S flag. /CMSH was added in Igor Pro 7.00.
/CSAF=*factor*	In rare situations that usually involve spatial degeneracy, the Voronoi interpolation algorithm may need additional memory. You can use the /CSAF flag with an integer factor greater than 1 to increase the memory allocation. /CSAF was added in Igor Pro 9.00.
/D=*splineDeg*	Specifies the spline degree with the Spline method. The default spline degree is 2. Supported values are 2, 3, 4, and 5.
/DEST=*destWave*	Specifies the wave to contain the output of the operation. If the specified wave already exists, it is overwritten. Creates a wave reference for the destination wave in a user function. See **Automatic Creation of WAVE References** on page IV-72 for details.
/E=*outerValue*	Assigns *outerValue* to all points outside the convex domain of the Delaunay triangulation. By default *outerValue* = NaN.
/F={*fx*,*fy*}	Calculates a bilinear interpolation of all the source data. Here *fx* is the sampling factor for the X-direction and *fy* is the sampling factor in the Y-direction. The output number of points in a dimension is factor*(number of data intervals) +1. The number of data intervals is one less than the number of points in that dimension. For example, if *srcWave* is a 2x2 matrix (you have a single data interval in each direction) and you use /F={2,2}, then the output wave is a 3x3 matrix (i.e., 2x2 intervals) which is a factor of 2 of the input. Sampling factors can be noninteger values.
/FDS	Performs spline interpolation to a "local" cubic spline that depends only on the 2D neighboring data. The interpolation goes exactly through the original data and provides continuity of the function and its first derivative. The second derivative is set to vanish on the boundary. The implemented algorithm was provided by Francis Dalaudier. /FDS was added in Igor Pro 7.00.
/FEQS	When performing Voronoi interpolation, forces the algorithm to use equal scaling for both axes. Normally the scaling for each axis is determined from its range. When the ranges of the two axes are different by an order of magnitude or more, it is helpful to force the larger range to be used for scaling both axes.

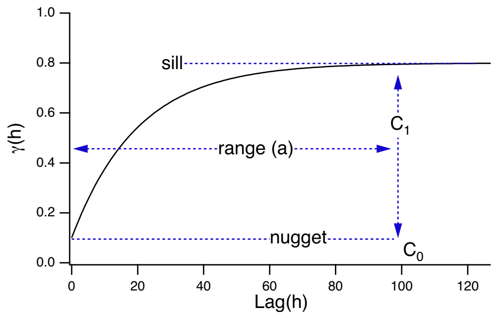
/FUNC=*funcName* Specifies the interpolation function. *funcName* can be:

nn	Nearest neighbor interpolation uses the value of the nearest neighbor without interpolation. This is the fastest function.
bilinear	Bilinear interpolation uses the immediately surrounding pixels and computes a linear interpolation in each dimension. This is the second fastest function.
cubic	Cubic polynomial (photoshop-like) uses a 4x4 neighborhood value to compute the sampled pixel value.
spline	Spline smoothed sampled value uses a 4x4 neighborhood around the pixel.
sinc	Slowest function using a 16x16 neighborhood.

/I=*iterations* Specifies the number of times the original triangulation is subdivided with the Voronoi interpolation method. By default the Voronoi interpolation computes the original triangulation without subdivision.

/K={*model*, *nugget*, *sill*, *range*}

Specifies the variogram parameters for kriging using standard notation, models are expressed in terms of the nugget value C_0, sill value C_0+C_1, and range *a*.



model Selects the variogram model. Values and models are:

1: Spherical. $\gamma(h) = C_0 + C_1 \cdot (3h/2a - 0.5 \cdot h^3/a^3)$

2: Exponential. $\gamma(h) = C_0 + C_1 \cdot (1 - \exp[-3 \cdot h/a])$

3: Gaussian. $\gamma(h) = C_0 + C_1 \cdot (1 - \exp[-3 \cdot (h/a)^2])$

nugget Specifies the lowest value in the variogram.

sill Specifies the maximum (plateau) value in the variogram range the characteristic length of the different variogram models.

Wave scaling has no effect on kriging calculations.

/PFTL=*tol* In Voronoi interpolation, controls the rectangular neighborhood about each datum position XY where the algorithm returns the original Z-value. The tolerance value 0 <= *tol* < 1 is tested separately in X and Y (i.e., it is not a geometric distance) when both X and Y are normalized to the range [0,1]. By default *tol*=1e-15.

Added in Igor Pro 7.00.

/PTW=*tWave* Uses a previous triangulation wave with Voronoi interpolation. *tWave* will be the wave saved by the /STW flag. You can't use a triangulation wave that was computed and saved on a different computer platform.

See **PTW Flag Example** on page V-388.

/PXSZ={*nx*,*ny*}	Specifies the size in pixels of the averaging rectangle used by the Pixelate operation. nx is the number of rows and ny is the number of columns that are averaged to yield a single output pixel. If srcWave is a 3D wave the resulting wave has the same number of layers as srcWave with pixelation computed on a layer-by-layer basis.
/PXSZ={*nx*,*ny*,*nz*}	Specifies the size in pixels of the averaging rectangle used by the Pixelate operation. nx is the number of rows and ny is the number of columns and nz is the number of layers that are averaged to yield a single output pixel. This form of the /PXSZ flag was added in Igor Pro 7.00.
/RESL={*nx*, *ny*}	Specifies resampling the full input image to an output image having *nx* rows by *ny* columns.

/S={*x0*,*dx*,*xn*,*y0*,*dy*,*yn*}

Calculates a bilinear interpolation of a subset of the source data. Here *x0* is the starting point in the X-direction, *dx* is the sampling increment, *xn* is the end point in the X-direction and the corresponding values for the Y-direction. If you set *x0* equal to *xn* the operation will compute the triangulation but not the interpolation.

/SPRT	Skips the XY perturbation step. The perturbation step is designed to break degeneracies that originate when the XY data are sampled on a rectangular grid. If your data are not sampled on a rectangular grid you can skip the perturbation and get better accuracy in reproducing the Z-values at the sampled locations. See also **ModifyContour** with the keyword Perturbation. Added in Igor Pro 7.00.
/STW	Saves the triangulation information in the wave W_TriangulationData in the current data folder. W_TriangulationData can only be used on the computer platform where it was created.
/SV	Saves the Voronoi interpolation in the 2D wave M_VoronoiEdges, which contains sequential edges of the Voronoi polygons. Edges are separated from each other by a row of NaNs. The outer most polygons share one or more edges with a large triangle containing the convex domain.

/TRNS={*transformFunc*,*p1*,*p2*,*p3*,*p4*}

Determines the mapping between a pixel in the destination image and the source pixel. *transformFunc* can be:

scaleShift	Sets image scaling which could be anamorphic if the X and Y scaling are different.
radialPoly	Corrects both color as well as barrel and pincushion distortion. In `radialPoly` the mapping from a destination pixel to a source pixel is a polynomial in the pixel's radius relative to the center of the image. A source pixel, *sr,* satisfies the equation: $$sr = ar + br^2 + cr^3 + dr^4,$$ where *r* is the radius of a destination pixel having an origin at the center of the destination image.

The corresponding parameters are:

transformFunc	***p1***	***p2***	***p3***	***p4***
scaleShift	xOffset	xScale	yOffset	yScale
radialPoly	a	b	c	d

/U=*uniformScale*	Calculates a bilinear interpolation of all the source data as with the /F flag but with two exceptions: A single uniform scale factor applies in both dimensions, and the scale factor applies to the number of points — not the intervals of the data.

/W={*xWave*, *yWave*}	Provides the scaling waves for XYWaves interpolation. Both waves must be monotonic and must have one more point than the corresponding dimension in *srcWave*. The waves contain values corresponding to the edges of data points in *srcWave*, so that the X value at the first data point is equal to `(xWave[0]+xWave[1])/2`.

Flags for Warp

/dgrx=*wave*	Sets the wave containing the destination grid X data.
/dgry=*wave*	Sets the wave containing the destination grid Y data.
/sgrx=*wave*	Sets the wave containing the source grid X data.
/sgry=*wave*	Sets the wave containing the source grid Y data.
/WM=*im*	Sets the interpolation method for warping an image. *im*=1: Fast selection of original data values. *im*=2: Linear interpolation. *im*=3: Smoothing interpolation (slow)

Details

When computing Bilinear or Spline interpolation *srcWave* can be a 2D or a 3D wave. When *srcWave* is a 3D wave the interpolation is computed on a layer by layer basis and the result is stored in a corresponding 3D wave. When the interpolation method is Kriging or Voronoi, *srcWave* is a 2D triplet wave (3-column wave) where each row specifies the X, Y, Z values of a datum. *srcWave* can be of any real data type. Results are stored in the wave M_InterpolatedImage. If *srcWave* is double precision so is M_InterpolatedImage; otherwise M_InterpolatedImage is a single precision wave.

Voronoi Interpolation Example

```
Function DemoVoronoiInterpolation()
    Make/O/N=(100,3) sampleTriplet
    sampleTriplet[][0]=enoise(5)
    sampleTriplet[][1]=enoise(5)
    sampleTriplet[][2]=sqrt(sampleTriplet[p][0]^2+sampleTriplet[p][1]^2)

    // Interpolate the data to a rectangular grid of 50x50 pixels
    ImageInterpolate/RESL={50,50}/DEST=firstImage voronoi sampleTriplet

    // Triangulate the XY locations and save the triangulation wave
    ImageInterpolate/STW voronoi sampleTriplet

    // Use the previous triangulation on the Z column of the sample
    MatrixOp/O zData=col(sampleTriplet,2)
    Wave W_TriangulationData
    ImageInterpolate/PTW=W_TriangulationData/RESL={50,50}/DEST=secondImage voronoi zData
End
```

PTW Flag Example

```
Function DemoPTW()
    // Create some random data
    Make/O/N=(100,3) eee = enoise(5)

    // Compute triangulation wave
    ImageInterpolate/RESL={1,1}/STW voronoi eee
    Wave W_TriangulationData

    // Copy the Z-column to a 1D wave
    MatrixOp/O e3 = col(eee,2)

    // Use the previous triangulation with 1D wave
    ImageInterpolate/PTW=W_TriangulationData /RESL={100,100} voronoi e3
    Wave M_InterpolatedImage
    Duplicate/O M_InterpolatedImage, oneD

    // Direct computation for comparison
    ImageInterpolate/RESL={100,100} voronoi eee
```