```
MatrixOp/O outWave=root:Images:peppers[][][0]
```

ImageTransform and MatrixOp are much faster for this size of image than the simple wave assignment. See **General Utilities: ImageTransform Operation** on page III-381 and **Using MatrixOp** on page III-140.

Interpolation and Sampling

You can use the **ImageInterpolate** operation (see page V-382) as both an interpolation and sampling tool. In the following example we create an interpolated image from a portion of the MRI image. The resulting image is sampled at four times the original resolution horizontally and twice vertically.

```
NewImage root:images:MRI
ImageInterpolate /S={70,0.25,170,70,0.5,120} bilinear root:images:MRI
NewImage M_InterpolatedImage
```

As the keyword suggests, the interpolation is bilinear. You can use the same operation to sample the image. In the following example we reduce the image size by a factor of 4:

```
NewImage root:images:MRI                // display for comparison
ImageInterpolate /f={0.5,0.5} bilinear root:images:MRI
NewImage M_InterpolatedImage            // the sampled image
```

Note that in reducing the size of certain images, it may be useful to apply a blurring operation first (e.g., MatrixFilter gauss). This becomes important when the image contains thin (smaller than sample size) horizontal or vertical lines.

If the bilinear interpolation does not satisfy your requirements you can use spline interpolations of degrees 2-5. Here is a comparison between the bilinear and spline interpolation of degree 5 used to scale an image:

```
ImageInterpolate /f={1.5,1.5} bilinear MRI
Rename M_InterpolatedImage Bilinear
NewImage Bilinear
ImageInterpolate /f={1.5,1.5}/D=5 spline MRI
NewImage M_InterpolatedImage
```

Another form of sampling is creating a pixelated image. A pixelated image is computed by subdividing the original image into non-overlapping rectangles of nx by ny pixels and computing the average pixel value for each rectangle:

```
ImageInterpolate/PXSZ={5,5}/DEST=pixelatedImage pixelate, MRI
NewImage pixelatedImage
```

Fast Fourier Transform

There are many books on the application of Fourier transforms in imaging so we will only discuss some of the technical aspects of using the **FFT** operation (see page V-222) in Igor.

It is important to keep in mind is that, for historical reasons, the default FFT operation *overwrites* and *modifies* the image wave. You can also specify a destination wave in the FFT operation and your source wave will be preserved. The second issue that you need to remember is that the transformed wave is converted into a complex data type and the number of points in the wave is also changed to accommodate this conversion. The third issue is that when performing the FFT operation on a real wave the result is a one-sided spectrum, i.e., you have to obtain the rest of the spectrum by reflecting and complex-conjugating the result.

A typical application of the FFT in image processing involves transforming a real wave of 2N rows by M columns. The complex result of the FFT is (N+1) rows by M columns. If the original image wave has wave scaling of dx and dy, the new wave scaling is set to 1/(N*dx) and 1/(M*dy) respectively.

The following examples illustrate a number of typical applications of the FFT in imaging.

Calculating Convolutions

To calculate convolutions using the FFT it is necessary that the source wave and the convolution kernel wave have the same dimensions (see **MatrixOp** convolve for an alternative). Consider, for example, smoothing noise via convolution with a Gaussian:

```
// Create and display a noisy image.
Duplicate /O root:images:MRI mri    // an unsigned byte image.
Redimension/s mri                   // convert to single precision.
mri+=gnoise(10)                     // add noise.
NewImage mri
ModifyImage mri ctab= {*,*,Rainbow,0}// show the noise using false color.

// Create the filter wave.
Duplicate/O mri gResponse     // just so that we have the same size wave.
SetScale/I x -1,1,"" gResponse
SetScale/I y -1,1,"" gResponse

// Change the width of the Gaussian below to set the amount of smoothing.
gResponse=exp(-(x^2+y^2)/0.001)

// Calculate the convolution.
Duplicate/O mri processedMri
FFT processedMri                    // Transform the source
FFT gResponse                       // Transform the kernel
processedMri*=gResponse    // (complex) multiplication in frequency space
IFFT processedMri

// Swap the IFFT to properly center the result.
ImageTransform swap processedMri
Newimage processedM
ModifyImage processedMri ctab= {*,*,Rainbow,0}
```

In practice one can perform the convolution with fewer commands. The example above has a number of commands that are designed to make it clearer. Also note that we used the **SetScale** operation (see page V-853) to create the Gaussian filter. This was done to make sure that the Gaussian was created at the center of the filter image, a choice that is compatible with the ImageTransform **swap** operation. This example is also not ideal because one can take advantage of the properties of the Gaussian (the Fourier transform of a Gaussian is also Gaussian) and perform the convolution as follows:

```
// Calculate the convolution.
Duplicate/O mri shortWay
FFT shortWay
shortWay*=cmplx(exp(-(x^2+y^2)/0.01),0)
IFFT shortWay
Newimage shortWay
ModifyImage shortWay ctab={*,*,Rainbow,0}
```

Spatial Frequency Filtering

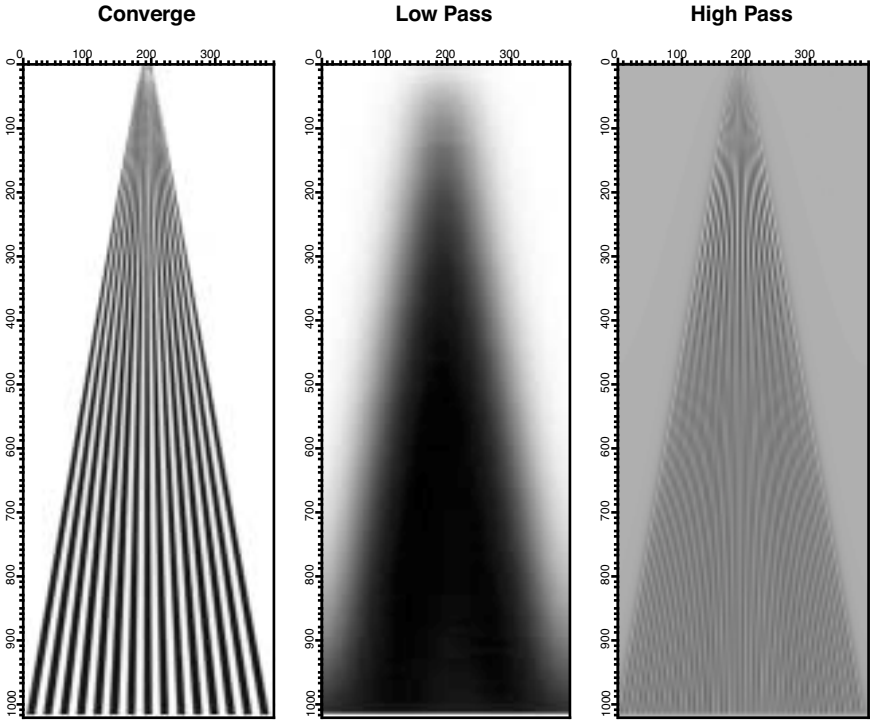
The concept behind spatial frequency filtering is to transform the data into spatial frequency space. Once in frequency domain we can modify the spatial frequency distribution of the image and then inverse-transform to obtain the modified image.

Here is an example of low and high pass filtering. The converge image consists of wide black lines converging to a single point. If you draw a horizontal line profile anywhere below the middle of the image you will get a series of 15 rectangles which will give rise to a broad range of spatial frequencies in the horizontal direction.

```
// Prepare for FFT; we need SP or DP wave.
Duplicate/O root:images:converge converge
Redimension /s converge
FFT converge
Duplicate/O converge lowPass     // new complex wave in freq. domain
lowPass=lowPass*cmplx(exp(-(p)^2/5),0)
```

```
IFFT lowPass
NewImage lowPass                    // nonoptimal lowpass

Duplicate/O converge hiPass
hiPass=hiPass*cmplx(1-1/(1+(p-20)^2/2000),0)
IFFT hiPass
NewImage hiPass                     // nonoptimal highpass
```



We arbitrarily chose the Gaussian form for the low-pass filter. In practical applications it is usually important to select an exact "cutoff" frequency and at the same time choose a filter that is sufficiently smooth so that it does not give rise to undesirable filtering artifacts such as ringing, etc. The high-pass filter that we used above is almost a notch filter that rejects low frequencies. Both filters are essentially one-dimensional filters.

Calculating Derivatives

Using the derivative property of Fourier transform, you can calculate, for example, the x-derivative of an image in the following way:

```
Duplicate/O root:images:mri xDerivative   // retain the original.
Redimension/S xDerivative
FFT xDerivative
xDerivative*=cmplx(0,p)    // neglecting 2pi factor & wave scaling.
IFFT xDerivative
NewImage xDerivative
```

Although this approach may not be appealing in all applications, its advantages are apparent when you need to calculate higher order derivatives. Also note that this approach does not take into account any wave scaling that may be associated with the rows or the columns.

Calculating Integrals or Sums

Another useful property of the Fourier transform is that the transform values along the axes correspond to integrals of the image. There is usually no advantage in using the FFT for this purpose. However, if the FFT is calculated anyway for some other purpose, one can make use of this property. A typical situation where this is useful is in calculating correlation coefficient (normalized cross-correlation).