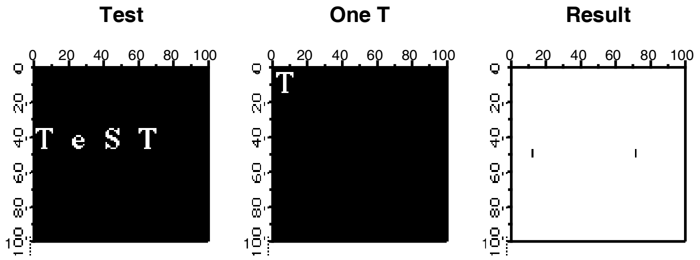
Correlations

The FFT can be used to locate objects of a particular size and shape in a given image. The following example is rather simple in that the test object has the same scale and rotation angle as the ones found in the image.

```
// Test image contains the word Test.
NewImage test                   // We will be looking for the two T's
Duplicate/O root:images:oneT oneT// the object we are looking for
NewImage oneT

Duplicate/O test testf          // because the FFT overwrites
FFT testf
Duplicate/O oneT oneTf
FFT oneTf
testf*=oneTf                    // not a "proper" correlation
IFFT testf
ImageThreshold/O/T=1.25e6 testf  // remove noise (due to overlap with other
characters
NewImage testf          // the results are the correlation spots for the T's
```



When using the FFT it is sometimes necessary to operate on the source image with one of the built-in window functions so that pixel values go smoothly to zero as you approach image boundaries. The **ImageWindow** operation (see page V-435) supports the Hanning, Hamming, Bartlett, Blackman, and Kaiser windows. Normally the **ImageWindow** operation (see page V-435) works directly on an image as in the following example:

```
// The redimension is required for the FFT operation anyway, so you
// might as well perform it here and reduce the quantization of the
// results in the ImageWindow operation.
Redimension/s blobs
ImageWindow /p=0.03 kaiser blobs
NewImage M_WindowedImage
```

To see what the window function looks like:

```
Redimension/S blobs                 // SP or DP waves are necessary
ImageWindow/i/p=0.01 kaiser blobs   // just creates the window data
NewImage M_WindowedImage      // you can also make a surface plot from this.
```

Wavelet Transform

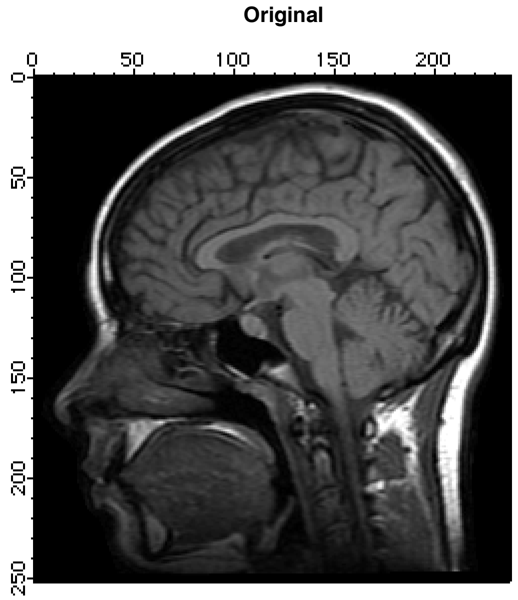
The wavelet transform is used primarily for smoothing, noise reduction and lossy compression. In all cases the procedure we follow is first to transform the image, then perform some operation on the transformed wave and finally calculate the inverse transform.

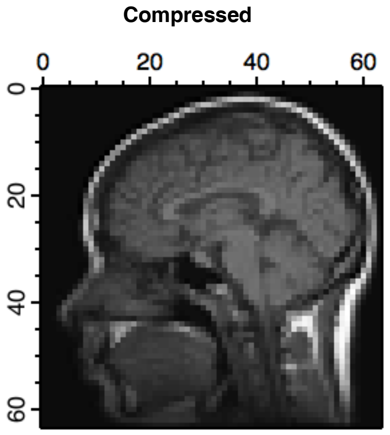
The next example illustrates a wavelet compression procedure. Start by calculating the wavelet transform of the image. Your choice of wavelet and coefficients can significantly affect compression quality. The compressed image is the part of the wave that corresponds to the low order coefficients in the transform (similar to low pass filtering in 2D Fourier transform). In this example we use the **ImageInterpolate** operation (see page V-382) to create a wave from a 64x64 portion of the transform.

```
DWT /N=4/P=1/T=1 root:images:MRI,wvl_MRI     // Wavelet transform
// reduce size by a factor of 16
ImageInterpolate/s={0,1,63,0,1,63} bilinear wvl_MRI
```

To reconstruct the image and evaluate compression quality, inverse-transform the compressed image and display the result:

```
DWT /I/N=4/P=0/T=1 M_InterpolatedImage,iwvl_compressed
NewImage iwvl_compressed
```





The reconstructed image exhibits a number of compression-related artifacts, but it is worth noting that unlike an FFT based low-pass filter, the advantage of the wavelet transform is that the image contains a fair amount of high spatial frequency content. The factor of 16 mentioned above is not entirely accurate because the original image was stored as a one byte per pixel while the compressed image consists of floating point values (so the true compression ratio is only 4).

To illustrate the application of the wavelet transform to denoising, we start by adding Gaussian distributed noise with standard deviation 10 to the MRI image:

```
Redimension/S Mri                       // SP so we can add bipolar noise
Mri+=gnoise(10)                         // Gaussian noise added
NewImage Mri
ModifyImage Mri ctab={*,*,Rainbow,0}//  false color for better discrimination.
DWT/D/N=20/P=1/T=1/V=0.5 Mri,dMri       // increase /V for more denoising
NewImage dMri                           // display denoised image
ModifyImage dMri ctab={*,*,Rainbow,0}
```