Igor supports eight built-in edge detectors (methods) that vary in performance depending on the source image. Some methods require that you provide several parameters which tend to have a significant effect on the quality of the result. In the following examples we illustrate the importance of these choices.

```
// Create and display a simple artificial edge image.
Make/B/U/N=(100,100) edgeImage
edgeImage=(p<50? 50:5)
Newimage edgeImage

// Try a simple Sobel detector using iterated threshold detection.
ImageEdgeDetection/M=1/N Sobel, edgeImage
NewImage M_ImageEdges
ModifyImage M_ImageEdges explicit=0 // to see binary image in color
ModifyImage M_ImageEdges ctab= {*,*,Rainbow,0}
```

This result (the red line) is pretty much what we would expect. Here are other examples that work similarly well:

```
ImageEdgeDetection/M=1/N Kirsch, edgeImage   // same output wave
```

or

```
ImageEdgeDetection/M=1/N Roberts, edgeImage  // same output wave
```

The innocent looking /M=1 flag implies that the operation uses an iterative automatic thresholding. This appears to work well in the examples above, but it fails completely when using the Frei filter:

```
ImageEdgeDetection/M=1/N Frei, edgeImage
```

On the other hand, the bimodal fit thresholding works much better here:

```
ImageEdgeDetection/M=2/N Frei, root:edgeImage
```

The performance of this filter improves dramatically if you add a little noise to the image:

```
edgeImage+=gnoise(1)
ImageEdgeDetection/M=1/N/S=1 Canny, edgeImage
```

Using More Exotic Edge Detectors

The more exotic edge detectors consist of multistep operations that usually involve smoothing and differentiation. Here is an example that illustrates the effect of smoothing:

```
Duplicate/O root:images:blobs blobs
ImageEdgeDetection/M=1/N/S=1 Canny,blobs
Duplicate/O M_ImageEdges smooth1
ImageEdgeDetection/M=1/N/S=2 Canny,blobs
Duplicate/O M_ImageEdges smooth2
ImageEdgeDetection/M=1/N/S=3 Canny,blobs
Duplicate/O M_ImageEdges smooth3
NewImage smooth1
ModifyImage smooth1 explicit=0
NewImage smooth2
ModifyImage smooth2 explicit=0
NewImage smooth3
ModifyImage smooth3 explicit=0
```

As you can see, the third image (smooth3) is indeed much cleaner than the first or the second, however, that result is obtained at the cost of loosing some of the small blobs. The following commands will draw a circle around one of the blobs that is missing in the third image:

```
DoWindow/F Graph0
SetDrawLayer UserFront
SetDrawEnv linefgc= (65280,0,0),fillpat= 0
DrawOval 0.29,0.41,0.35,0.48
DoWindow/F Graph1
SetDrawLayer UserFront
SetDrawEnv linefgc= (65280,0,0),fillpat= 0
```

```
DrawOval 0.29,0.41,0.35,0.48
DoWindow/F Graph2
SetDrawLayer UserFront
SetDrawEnv linefgc= (65280,0,0),fillpat= 0
DrawOval 0.29,0.41,0.35,0.48
```

It is instructive to make a similar set of images using the Marr and Shen detectors.

```
// Note: This will take considerably longer time to execute!
Duplicate/O root:images:blobs blobs
ImageEdgeDetection/M=1/N/S=1 Marr,blobs
Duplicate/O M_ImageEdges smooth1
ImageEdgeDetection/M=1/N/S=2 Marr,blobs
Duplicate/O M_ImageEdges smooth2
ImageEdgeDetection/M=1/N/S=3 Marr,blobs
Duplicate/O M_ImageEdges smooth3
NewImage smooth1
ModifyImage smooth1 explicit=0
NewImage smooth2
ModifyImage smooth2 explicit=0
NewImage smooth3
ModifyImage smooth3 explicit=0
SetDrawLayer UserFront
SetDrawEnv linefgc= (65280,0,0),fillpat= 0
DrawOval 0.29,0.41,0.35,0.48
```

The three images of the calculated edges demonstrate the reduction of noise with the increase in the size of the convolution kernel. It’s also worth noting that the blob that disappeared when we used the Canny detector is clearly visible using the Marr detector.

In the following example we use the Shen-Castan detector with various smoothing factors. Note that this edge detection algorithm does not use the standard thresholding (you have to specify the threshold using the /F flag).

```
Duplicate/O root:images:blobs blobs
ImageEdgeDetection/N/S=0.5 shen,blobs
Duplicate/O M_ImageEdges smooth1
ImageEdgeDetection/N/S=0.75 shen,blobs
Duplicate/O M_ImageEdges smooth2
ImageEdgeDetection/N/S=0.95 shen,blobs
Duplicate/O M_ImageEdges smooth3
NewImage smooth1
ModifyImage smooth1 explicit=0
NewImage smooth2
ModifyImage smooth2 explicit=0
NewImage smooth3
ModifyImage smooth3 explicit=0
SetDrawLayer UserFront
SetDrawEnv linefgc=(65280,0,0),fillpat=0
DrawOval 0.29,0.41,0.35,0.48
```

As you can see in this example, the Shen detector produces a thin, though sometimes broken, boundary. The noise reduction is a trade-off with edge quality.

One of the problems of edge detectors that employ smoothing is that they usually introduce errors when there are two edges that are relatively close to each other. In the following example we construct an artificial image that illustrates this point:

```
Make/B/U/O/N=(100,100) sampleEdge=0
sampleEdge[][49]=255
sampleEdge[][51]=255
NewImage sampleEdge
ImageEdgeDetection/N/S=1 Marr, sampleEdge
Duplicate/O M_ImageEdges s2
```