NAME

angle − gradient direction of edges

SYNOPSIS

theta = angle(DeltaX, DeltaY)

PARAMETERS

DeltaX and Delta Y

M x N intensity images, resulting from convolution with gradient masks in two orthogonal directions

theta Intensity image. Each pixel has the angle of the approximated gradient, in radians. Where DeltaX is zero, theta is taken as PI/2.

DESCRIPTION

The function **angle** obtains the gradient direction "theta" from images yielded from convolution with gradient masks in direction x and y (Deltax and Deltay, respectively).

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

Img = imread('disks.bmp');
xbasc()
imshow(Img,2);
Dx = edge(Img,thresh=-1, direction='horizontal');
Dy = edge(Img,thresh=-1, direction='vertical');
t = angle(Dx,Dy);
imshow(t,[])

chdir(initial_dir);
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

edge

NAME

bwborder − border detection for binary images

SYNOPSIS

B = bwborder(Img)
B = bwborder(Img, 4)
B = bwborder(Img, 8)

PARAMETERS

Img binary array, 1 for object and 0 for background (double precision)

B binary(0-1) array (double), same size as Img

DESCRIPTION

Extracts contours from binary images, by detecting which pixel valued 1 has at least one neighbor valued 0. The second argument is 4 or 8 depending if the detection is to be made in 4- or 8-neighborhood. The final border will be 8-connected if 4-neighbors are used, and 4-connected if 8-neighbors are used.

bwborder(Img) equals **bwborder(Img, 8)**.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

Img = imread('bin2.pbm')
xbasc()
imshow(Img,2);
B = bwborder(Img)
imshow(B,2);

chdir(initial_dir);
```

BUGS AND SHORTCOMINGS

Images are stored in double precision matrices. Hopefully, the next release will make usage of integer types.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

edge, follow, im2bw

NAME

bwdist − distance transforms

SYNOPSIS

dt = bwdist(img [,algorithm])

PARAMETERS

img Binary image containing one or more binary shapes. (foreground == 1, background == 0). The shapes may have any form.

algorithm (see references)

Listed below are the various algorithms available, together with the shortest string form accepted (for speed of use). This argument is CASE-INSENSITIVE. Some of the algorithms are faster than others, but this depends heavily on the size and content of the input.

´euclidean´ or ´euclid´: the default fast exact euclidean algorithm. Currently, it is the same as the ´cuisenaire pmn´ bellow. (DEFAULT)

´cuisenaire pmn´: very fast exact euclidean algorithm. It is based on propagation of multiple neighborhoods to build up an exact EDT.

´cuisenaire pmon´: a variation of the latter that uses multiple oriented neighborhoods. It seems to be slightly slower, in general, but can be faster for some cases.

´cuisenaire psn4´: a variation of the latter that uses only 4-neighborhood. This is faster but less precise.

´cuisenaire psn8´: a variation of the latter that uses diagonal neighborhood after 4-neighborhood to improve the precision. This is faster than the full ´pmn´ algorithm, but less precise. It is a little slower than psn4 but considerably more precise.

´maurer´ or ´mau´ : very fast (and recent) exact euclidean algorithm, based on some dimensionality properties of Voronoi diagrams. Seems to be slightly slower than ´cuisenaire pmn´, but can be faster for some cases.

´lotufo-zampirolli´ or ´lotufo-z´: very fast exact euclidean algorithm. Seems to be slightly slower than maurer and cuisenaire, in general, but can be faster for some cases.

´IFT 8´ or ´IFT´: a fast algorithm using the euclidean metric. For large and thick shapes, there may be a few small errors, which are dispensable for most practical applications.

´IFT 4´: the same algorithm but with 4-neighborhood propagation. This means that this method is about 2x faster but less precise.

´exact dilations´ or ´exact dil´: will perform an exact euclidean algorithm that is slow for medium shapes, but it is always exact and reasonably fast for thin shapes.

dt The distance transform of the image. When using the euclidean metric, it has the squared euclidean distances of any point of the image to the boundary of the object.

DESCRIPTION

Function **bwdist** computes the distance transform. For each foreground pixel (i.e. value ´1´) in the input image, the distance transform assigns a value that is the smallest distance between that pixel and the outer boundary of the object.

Many different methods are provided for comparison purposes. If you are going to use bwdist extensively, you could test the algorithms to find the best one for your particular type of image.

EXAMPLE

```
xset('auto clear', 'on');

// First, a simple example to illustrate the concept
A = zeros(15,10);
A(4:12,3:7)=1;
A(4:5,3:4)=0

D = bwdist(A)
D = sqrt(D)
// Note how the values in D were calculated.
// For each pixel p such that A(p)=1, D(p) is the minimum euclidean
// distance of p and the 0-pixels (background).


// ------------------------------------
//  Now to a more interesting example
// ------------------------------------


A = gray_imread(SIPDIR + 'images/escher.png');
imshow(A,2);

D = bwdist(A);        // method=='cuisenaire pmn'
imshow(log(1+D),[]); // normalizes image to enhance visualization

D = bwdist(A,'exact dilations');
imshow(log(1+D),[]);

// To obtain an external EDT, simply invert the shape:
B = 1-A;
D = bwdist(B,'maurer');
imshow(log(1+D),[]);

// To obtain an external+internal EDT, simply compute
// the binary border of the shape and pass its negative
// to bwdist:
A = bwborder(A);
A = 1-A;
D = bwdist(A,'lotufo-zampirolli');
imshow(log(1+D),[]);

// ---------------------------------
//  Other forms to visualize the DT
// ---------------------------------

// Wrapping (note the wavefronts of iso-distance)
imshow(modulo(sqrt(D),10),[])

// Usual:
D = bwdist(A);
```

```
D = normal(sqrt(D),1000,1);
imshow(D,hotcolormap(1000));

// There is also of DT application in the example for the "watershed"
// function.

xset('auto clear', 'off');
```

REFERENCES

´Cuisenaire´:
Cuisenaire, O and Macq, B, "Fast Euclidean Distance Transformation by Propagation Using Multiple Neighborhoods", Computer Vision and Image Understanding, no. 2, vol 76, 163--172, 76, 1999.

Chapter 3 of "Distance transformations: fast algorithms and applications to medical image processing", Olivier Cuisenaire's Ph.D. Thesis, October 1999, Université catholique de Louvain, Belgium.

´Maurer´:
Maurer, C.R. and R. Qi and V. Raghavan, "A Linear Time Algorithm for Computing the Euclidean Distance Transform in Arbitrary Dimensions", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 2, pp. 265-270, february 2003.

´IFT´:
"Multiscale Skeletons by Image Foresting Transform and its Application to Neuromorphometry", A.X. Falcao, L. da F. Costa, B.S. da Cunha, Pattern Recognition, 2002.

´Lotufo-Zampirolli´:
R. Lotufo and F. Zampirolli, Fastmultidimensional parallel euclidean distance transform based on mathematical morphology, in T. Wu and D. Borges, editors, Proccedings of SIBGRAPI 2001, XIV Brazilian Symposium on Computer Graphics and Image Processing, pages 100-105. IEEE Computer Society, 2001.

´Exact Dilations´:
"Multiresolution shape representation without border shifting", L. da F. Costa, and L. F. Estrozi, Electronics Letters, no. 21, vol. 35, pp. 1829-1830, 1999.

"Shape Analysis and Classification", L. da F. Costa and R.M. Cesar Jr., CRC Press.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

skel, thin

NAME

bwlabel − connected component labeling

SYNOPSIS

[L, n] = bwlabel(img [, nhood])

PARAMETERS

img A binary image, where 0 stands for background.

nhood

A scalar. The connectivity to consider in the algorithm. May be 4 or 8. Defaults to 8.

L A matrix of the same size as img, with the pixels of each connected object having the same number. The numbers vary from 1 to N, where N is the number of connected objects. The background is numbered 0.

n The number of connected components. Equals to maxi(L).

DESCRIPTION

Function **bwlabel** numbers all the objects in a binary image. One common application is to filter out objects that have less than a certain ammount of pixels. See the examples.

You can use the Scilab **find** function in conjunction with **bwlabel** to return vectors of indices for the pixels that make up a specific object. For example, to return the coordinates for the pixels in object 3,

```
[r,c] = find(bwlabel(BW)==3)
```

EXAMPLE 1

```
Img =[0   0   0   0   0   1   1
      0   1   1   0   0   1   1
      0   1   1   0   0   1   1
      0   0   0   1   0   1   1
      0   0   0   1   0   1   1
      0   0   0   1   0   1   1
      0   0   1   1   0   1   1
      0   0   0   0   0   1   1];

L = bwlabel(Img,4)

// Objects 2 and 3 are connected if 8-connectivity is used:

L = bwlabel(Img) // default: 8-connectivity

[r,c] = find(L==2);

rc = [r' c']    // coordinates of object 2!
```

EXAMPLE 2

```
xset('auto clear', 'on');

a = gray_imread(SIPDIR + 'images/disks.bmp');
```

```
// Add some noise
//
a = imnoise(a,'salt & pepper');
a = 1-a;
imshow(a,2);  // convention: objects are white(1)

// Label every connected component with a unique number.
//
[L, n] = bwlabel(a);

// Shows each component with a different color
//
imshow(L+1, rand(n+1,3));

// Get one specific region (probably a single noise point)
reg = (L == 300);
imshow(reg*1, 2);

// Eliminate regions smaller than 100 pixels (noise)
// and those larger than 1000 pixels (cluttered disks)
for i=1:n
  f = find(L==i);      // linear coordinates of i-th region
  reg_size = size(f,'*');
  if reg_size < 100 | reg_size > 1000
    L(f) = 0;          // merge small regions with the background
  end
end

imshow(L+1, rand(n+1,3));   // note how the small regions are gone


// Just as a side-activity, let's fill the unwanted holes:

bw = 1*(L>0);  // binarize the image
imshow(bw,2)
bw = dilate(bw);
bw = erode(bw);
imshow(bw,2);  // every hole is now filled

xset('auto clear', 'off');
```

REFERENCES

We use a simple stack-based flooding implementation written in C, but there exist many faster algorithms.

The flood/fill region growing process may be found in most books of imaging science. For instance:
"Shape Analysis and Classification", L. da F. Costa and R. M. Cesar Jr., CRC Press, pp. 335-347.

Example of fast algorithm (not implemented):
Haralick, Robert M., and Linda G. Shapiro, Computer and Robot Vision, Volume I, Addison-Wesley, 1992, pp. 28-48.

REMARKS

Images cannot have more than 2^32 pixels. For example, we do not accept images larger than 65535x65535

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

bwborder, erode, dilate, watershed

NAME

curvature2d − curvature of a surface or image

SYNOPSIS

Output = curvature2d(Input)

Input 2D matrix (e.g. intensity image)

Output

2D matrix, the estimated curvature function

DESCRIPTION

"curvature2d" calculates the curvature of an intensity image or surface represented as a 2D matrix. It currently uses finite differences.

AUTHORS

Leandro Estrozi <lfestrozi@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

curvature, minmax, fftderiv

NAME

curvature − curvature of a contour

SYNOPSIS

K = curvature(BW [,sigma, delta])
K = curvature(x,y [,sigma, delta])

PARAMETERS

K vector containing the curvature of the contour at each point.

BW Binary image containing only one object (0 for background, 1 for object).

x and y
vectors, storing the parametrized contour.

sigma standard deviation of the gaussian function used to smooth the contour before computing the curvature. Defaults to 5.

delta a double number, the time between samples (delta t), and defaults to 1.

DESCRIPTION

Function **curvature** calculates the curvature at each point of a binary contour, using FFT and a formula from differential geometry.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

Img = imread('star.bmp');
xbasc()
imshow(Img,2);
k = curvature(Img,13);  // 13 sigma (shape is smoothed so curvature exists)
xbasc()
plot(k)
//
// observe there are six curvature peaks,
// corresponding to the six peaks of the star. There
// is one peak half at 0 and half at about 450.
// That's because the parametrization of
// the contour started at the highest peak and
// terminated there. Note also that the shape had to
// be considerably smoothed so the curvature doesn't
// blow up at the very sharp peaks of the star.
//

chdir(initial_dir);
```

REFERENCES

"Shape Analysis and Classification", L. da F. Costa and R. M. Cesar Jr., CRC Press, pp. 335-347.

"Differential Geometry of Curves and Surfaces", Manfredo P. do Carmo, Prentice Hall, 1976.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

follow, gsm, fftderiv

NAME

dilate − morphological dilation of binary images

SYNOPSIS

E = dilate(Img, [SE, center])

PARAMETERS

Img M x N Binary image to be dilated. (0 for background, 1 for object)

E M x N Binary dilated image.

SE Arbitrary structuring element represented as a binary array. Defaults to:

```
[0 1 0
 1 1 1
 0 1 0]
```

center origin of structuring element. Shold be within image dimensions. Defaults to the center of the SE array.

DESCRIPTION

Function **dilate** performs morphological dilation of a binary image Img using *SE* as the structuring element.

EXAMPLE

```
Img = imread(SIPDIR+'images/tru.jpg');
Img = 1-im2bw(Img, 0.3);
Img = thin(Img);
xbasc()
imshow(Img ,2);

e = dilate(Img);
xbasc()
imshow(e ,2);

SE = eye(10,10);
e = dilate(Img, SE, [1,1]);
xbasc()
imshow(e ,2);
```

REMARKS

The algorithm is fully functional, but there exists many better ones. The present implementation will certainly change, but the interface shall remain unaltered.

REFERENCE

"Morphological Algorithms", Luc Vincent, in "Mathematical Morphology in Image Processing", Ed. Marcel Dekker, 1993.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

erode, edilate, bwdist, watershed (example)

NAME

drawline − draws line in a binary image

SYNOPSIS

imo = drawline(img, points)

PARAMETERS

img Intensity image

points n rows x 2 columns vector of n 2D (row,col) coordinates:
[row1 col1; row2 col2; ... ; rowN colN]

DESCRIPTION

Function **drawline** is used to draw a digital straight line or polyline into an image.

EXAMPLE

```
I = zeros(100,100);
J = drawline(I, [1 1; 20 50; 100 100]);
xbasc();
imshow(J,2);
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

mogrify (its -draw parameter provides many shapes with antialiasing)

NAME

edge − edge detection

SYNOPSIS

E = edge(Img)
E = edge(Img, named_args)
E = edge(Img, method)
E = edge(Img, method, thresh)
E = edge(Img, method, thresh, direction)
E = edge(Img, method, thresh, direction, sigma)

PARAMETERS

Img M x N Grayscale (intensity) image in any range.

method
may be **'sobel'**(default), **'prewitt'** or **'fftderiv'. Other methods will appear in the future.**

thresh
sets the threshold level, from **0 to 1**. Defaults to 0.5. If negative, then the output image, *E*, will have the un-thresholded gradient image.

direction
may be **'horizontal'**, **'vertical'** or **'both'**(default). This determines the direction to compute the image gradient.

sigma Controls the ammount of high-frequency attenuation in some methods (only the 'fftderiv' method uses this parameter). This can be used to obtain different levels of detail and to filter out high-frequency noise. Defaults to 1.

<**named_args**>
this is a sequence of statements key1=value1, key2=value2,... where key1, key2,... can be any of the optional arguments above, in any order.

DESCRIPTION

The function **edge** performs edge detection on a grayscale intensity image. The user may set the method, the threshold level and the direction of the edge detection.

edge(Img) Detects edges in *Img*, using the sobel gradient estimator, 0.5 threshold level and in both horizontal and vertical directions.

The other parameters are optional and non-positional. That is, they may be passed to the function by their name. The following example illustrates this.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

Img = imread('tru.jpg');
Img = im2gray(Img);
xbasc()
imshow(Img);

e = edge(Img);  // sobel, thresh = 0.5
xbasc()
imshow(e,2)

e = edge(Img,'prewitt'); // thresh = 0.5
xbasc()
imshow(e,2)
```

```
e = edge(Img,'fftderiv', 0.4);  // FFT gradient method; 0.4 threshold
xbasc()
imshow(e,[])

// It is useful to thin the edges, eliminating redundant pixels:
e = thin(e);
xbasc()
imshow(e,[])

e = edge(Img,'fftderiv',sigma=3,thresh=-1); // thicker edges, no threshold
xbasc()
imshow(e,[])

e = edge(Img,thresh=-1);
xbasc()
imshow(e,[])

chdir(initial_dir);
```

REMARKS

In the future, more sophisticated algorithms like Canny and Marr-Hildreth will be available, while maintaining the same interface.

REFERENCES

"Shape Analysis and Classification", L. da F. Costa and R. M. Cesar Jr., CRC Press, section 3.3.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

bwborder, mogrify, im2gray, imconv, mkfilter

NAME

edilate − euclidean morphological dilation of binary images

SYNOPSIS

Dil = edilate(Img, [radius, form])

PARAMETERS

Img M x N Binary image to be dilated. (0 for background, 1 for object)

radius - of circular euclidean structuring element to be used. Defaults to 5.

form determines the size of Dil. It assumes be one of the following values:

'same'
Dil has the same size as the original image, M x N. The image is assumed to be 0 outside its bounds.

'full' Dil contains the full dilation, (M+2*radius-1) x (N+2*radius-1). The image is assumed to be 0 outside its bounds.

DESCRIPTION

Function **edilate** performs exact euclidean dilation of a binary image Img using a circle as the structuring element. It is implemented in C to be fast.

EXAMPLE

```
Img = gray_imread(SIPDIR+'images/tru.jpg');
Img = 1-im2bw(Img, 0.3);
xbasc()
imshow(Img,2);
e = edilate(Img,7);
xbasc()
imshow(e,2);
```

REFERENCES

"Shape Analysis and Classification", L. da F. Costa and R. M. Cesar Jr., CRC Press.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

dilate, erode, bwdist, thin, skel

NAME

erode − morphological erosion of binary images

SYNOPSIS

E = erode(Img, [SE, center])

PARAMETERS

Img M x N Binary image to be eroded. (0 for background, 1 for object)

E M x N Binary eroded image.

SE Arbitrary structuring element represented as a binary array. Defaults to:

```
[0 1 0
 1 1 1
 0 1 0]
```

center origin of structuring element. Shold be within image dimensions. Defaults to the center of the SE array.

DESCRIPTION

Function **erode** performs morphological erosion of a binary image Img using *SE* as the structuring element.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

Img = gray_imread('disks2.bmp');
imshow(Img,2);
Img = 1-Img;
SE = ones(10,10);
e = erode(Img, SE);
xbasc()
imshow(e ,2);

chdir(initial_dir);
```

REMARKS

The algorithm is fully functional, but there exists many better ones. The present implementation will certainly change, but the interface shall remain unaltered.

REFERENCE

"Morphological Algorithms", Luc Vincent, in "Mathematical Morphology in Image Processing", Ed. Marcel Dekker, 1993.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

dilate, edilate, bwdist

NAME

fftderiv − derivative of a vector using FFT

SYNOPSIS

[Dy, DY] = fftderiv(y, [n , sigma, delta, in, out])
[Dy, DY] = fftderiv(y, <named_args>)

INPUT PARAMETERS

y vector containing a periodic window of a function to be differentiated.

n the order of the derivative. It is 1 for 1st derivative, 2 for 2nd derivative, and so on.

sigma the standard deviation of the gaussian kernel used to smooth the input. If *sigma* is zero, **fftderiv** will not smooth the input. (Defaults to 5)

delta a double number. If the input is in the time domain, this is the time between samples (delta t), and defaults to 1. If the input is in the frequency domain, this is the frequency increment between samples (delta f), and defaults to 1/N, where N is the number of samples.

in indicates if the input, *x*, is a function of time (no FFT has been applied) or frequency (FFT has already been applied). Can be 'time' or 'frequency'. (Defaults to 'time')

out indicates if the output, *xsm*, is a function of time (inverse FFT will be applied) or frequency (inverse FFT will not be applied). Can be 'time' or 'frequency'. (Defaults to 'time')

<**named_args**>
This is a sequence of statements key1=value1, key2=value2,... where key1, key2,... can be any of the optional arguments above, in any order.

OUTPUT PARAMETERS

Dy the derivative vector in "time" or "frequency" domain.

FDy the derivative vector in "frequency" domain.

DESCRIPTION

Function **fftderiv** performs the *n*-th derivative of a periodic function, stored in a vector, using FFT. The optional arguments *in* and *out* enables the user to reuse previously done FFTs. Here are some possible uses of gsm:

Dy = fftderiv(y)
n defaults to 1, sigma defaults to 5, *in* and *out* both defaults to 'time'.

Dy = fftderiv(y,2,3)
n equals 2, *sigma* equals 3, *in* and *out* both defaults to 'time'.

Dy = fftderiv(y,sigma=3, in='frequency',out='frequency')
n dafaults to 1, *y* in frequency domain (fft has already been done). *Dy* in frequency domain (inverse fft is NOT done by gsm)

In all above examples, *FDy* is in the frequency domain. It is the second output parameter, and thus it was discarded in the above examples.

EXAMPLE

```
step = 2*%pi/100;
y = sin(step:step:2*%pi);  // from 2pi/100 to 2pi
xbasc()
plot(y);
// 1st derivative, a sigma of 3 steps to the left and to the right
d = fftderiv(y,1,3*step,step);
xbasc()
plot(d)  // a cosine period
```

REMARKS

For a derivative without noises, the vector y must be an exact period of a continuous periodic function, i.e., its repetition has to be continuous. A direct way for checking this is to plot z = [y y] and look close in the middle. If there is not a minimal discontinuity, then **fftderiv** will certainly work without need for smoothing.

y should be smoothed before using **fftderiv** so the derivative is less sensitive to discontinuities and aliasing. For an estimation of the **sigma** parameter, please refer to the references below.

REFERENCES

"Shape Analysis and Classification", L. da F. Costa and R. M Cesar Jr., CRC Press, pp. 335-347.

"1D and 2D Fourier-based approaches to numeric curvature estimation and their comparative performance assessment", L. F. Estrozi, L. G. R. Filho, A. G. Campos and L. da F. Costa, Digital Signal Processing, 2002, accepted paper.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

follow, gsm, fftshift, curvature

NAME

follow − a contour follower

SYNOPSIS

[x,y] = follow(Img)
[x,y] = follow(Img, 4)
[x,y] = follow(Img, 8)

PARAMETERS

Img binary array, 1 for object and 0 for background (double precision)

x and y
vectors, storing the parametrized contour.

DESCRIPTION

Function **follow** extracts parametric contours of binary objects. This is useful for further extracting object features such as curvature and bending energy.

It is assumed that *Img* has only one object.

x and *y* both store the parametrized contour. That is, (x(i),y(i)) is a point of the contour, where the coordinate system is assumed as starting from bottom-left corner (0,0) to uper-right corner of the image. To get the (row,col) matrix coordinates, use the transformation below:

```
row = size(Img,'r') - y
col = x+1
```

The second argument is an optional parameter, being 4 or 8 depending if the border following is to be made in 4- or 8-neighborhood sequence, yielding a 4- or 8-connected contour.

follow(Img) equals **follow(Img, 8)**.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

Img = imread('star.bmp');
xset('window',0)
xbasc()
imshow(Img,2);
[x,y] = follow(Img);
xset('window',1)
xbasc()
plot2d(1:size(x,'*'),x,2);
plot2d(1:size(y,'*'),y,1);

chdir(initial_dir);
```

REFERENCE

"Shape Analysis and Classification", L. da F. Costa and R. M Cesar Jr., CRC Press, pp. 335-347.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

unfollow, gsm, bwborder, curvature

NAME

fractal − multiscale fractal dimension curve

DESCRIPTION

"fractal" routine is under development. It's sourcecode has some documentation.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

other toolboxes

NAME

gray_imread − read any image as grayscale

SYNOPSIS

im = gray_imread(filename)

PARAMETERS

Img 2D array representing pixel intensities

DESCRIPTION

This is a simple utility routine to read any image as grayscale. If the image is truecolor or pseudo-color, it is converted using im2gray

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

Img = gray_imread('ararauna.png');
xbasc()
imshow(Img,[]);

chdir(initial_dir);
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

imshow, im2gray, im2bw

NAME

gsm2d − 2D gaussian smoothing

SYNOPSIS

MG = gsm(M [,sigma]);

PARAMETERS

M the matrix (intensity image) to be smoothed

sigma the standard deviation of the gaussian kernel. Defaults to 5.

MG the smoothed image.

DESCRIPTION

Function **gsm** performs 2D gaussian smoothing of the image M, with standard deviation sigma, using FFT.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

[Img,map] = imread('onca.gif');
xbasc()
imshow(Img,map);
Img = im2gray(Img, map);   // Img is now 0-1 range
xbasc()
imshow(Img);
Img = gsm2d(Img,2);
xbasc()
imshow(Img);

chdir(initial_dir);
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

gsm, imconv, fft

NAME

gsm − 1D gaussian smoothing

SYNOPSIS

[xsm, Xsm] = gsm(x [, sigma, delta, in, out])
[xsm, Xsm] = gsm(x, <named_args>)

INPUT PARAMETERS

x the vector to be smoothed (row or column vector), real or complex.

sigma the standard deviation of the gaussian kernel. If *sigma* is zero, **gsm** returns the input vector unaltered in xsm. (Defaults to 5)

delta a double number. If the input is in the time domain, this is the time between samples (delta t), and defaults to 1. If the input is in the frequency domain, this is the frequency increment between samples (delta f), and defaults to 1/N, where N is the number of samples.

in indicates if the input, *x*, is a function of time (no FFT has been applied) or frequency (FFT has already been applied). Can be 'time' or 'frequency'. (Defaults to 'time')

out indicates if the output, *xsm*, is a function of time (inverse FFT will be applied) or frequency (inverse FFT will not be applied). Can be 'time' or 'frequency'. (Defaults to 'time')

<named_args>
This is a sequence of statements key1=value1, key2=value2,... where key1, key2,... can be any of the optional arguments above (*sigma, in, out*), in any order.

OUTPUT PARAMETERS

xsm the smoothed vector in "time" or "frequency" domain.

Xsm the smoothed vector in "frequency" domain.

DESCRIPTION

Function **gsm** performs gaussian smoothing of the vector *x*, with standard deviation *sigma*, using FFT. The optional arguments *in* and *out* enables the user to reuse previously done FFTs. Here are some possible uses of gsm:

xsm = gsm(x)
sigma defaults to 5, *in* and *out* both defaults to 'time'.

xsm = gsm(x,15)
sigma equals 15, *in* and *out* both defaults to 'time'.

xsm = gsm(x,15, out='frequency')
x in time domain. *xsm* in frequency domain (inverse fft is NOT done by gsm)

xsm = gsm(x,15,in='frequency', out='frequency')
x in frequency domain (fft has already been done). *xsm* in frequency domain (inverse fft is NOT done by gsm)

xsm = gsm(x, 'frequency', delta=0.1)
delta f is 0.1 *sigma* defaults to 5 *x* in frequency domain. *xsm* in time domain (inverse fft is done by gsm)

In all above examples, *Xsm* is in the frequency domain. It is the second output parameter, and thus it was discarded in the above examples.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');
```

```
Img = imread('star.bmp');
xset('window',0);
xbasc()
imshow(Img,2);
[x,y] = follow(Img);  // get the parametric contour
xset('window',1)
xbasc()
t=1:size(x,'*');
plot2d(t,x,2);
plot2d(t,y,1);
xsm = gsm(x,15);  // gaussian-smooth the contour
ysm = gsm(y,15);
// builds an image from parametric contour:
Img2=unfollow(xsm,ysm,size(Img));
xset('window',0);
xbasc()
imshow(Img2,2);

chdir(initial_dir);
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

gsm2d, fft

NAME

hello_sip − learn to add your own C routine to SIP

SYNOPSIS

sum = hello_sip(a,b)

PARAMETERS

a double-precision scalar

b double-precision scalar

c double-precision scalar

DESCRIPTION

"hello_sip" is a very simple example routine. The aim is that new developers know quickly how to add a new C-language implemented function to SIP.

hello_sip(a,b) simply returns a + b through a C routine.

The following files in the SIP sourcecode tree are related to the implementation of hello_sip:

src/hello_int.c --
interfaces C to Scilab

src/builder.sce --
specifies the interface files and scilab name; this script is ran by Scilab to compile the interface C routines into a shared library.

From AnImaL (aminal.sourceforge.net),
animal/hello.c -- routine that returns a+b

From AnImaL (aminal.sourceforge.net),
animal/hello.h -- header with the prototype of hello

From AnImaL (aminal.sourceforge.net),
animal/animal.h general header for the internal C library

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

In the SIP source tree, start with the "hello.c" file inside the directory "src".

Obsoleted: Figure 1 of the first SIP monograph (written in portuguese), which may be found in the SIP home page. SIP now uses an independent computer vision library, AnImaL, for its internal processing. See: http://animal.sourceforge.net

Moreover, in the scilab source tree, the directories "examples/interface-tour.so" and "examples/interface-tutorial.so" provide detailed information on interfacing C and Scilab.

NAME

hough − hough transform for line detection

SYNOPSIS

[ht, rho_range] = hough(imbin)

PARAMETERS

imbin binary image array (foreground equals 1)

ht Hough transform accumulation matrix. Rho varies along rows; theta varies along columns. Theta range is 0-179 degrees w/ 180 samples.

rho_range
Vector containing the range of radius values. Equals -rmax:rmax, where rmax is the maximum radius possibly found in the input image.

DESCRIPTION

Function **hough** calculates the hough transform of a binary image. The coordinate system is centered in the image, and the Y axis points downwards. Theta grows from X axis to Y axis. Negatives rho's point to the upper half of the image.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

// ======= Example 1

im = imread('star.bmp');
im = bwborder(im);
xbasc()
imshow(im,2);

h = hough(im);
xbasc()
imshow(h,[]);  // theta varies horizontally from 0 to 180

ht = 1*(h>= 40); // threshold the hough space
lim = ihough(ht,size(im)); // draw the detected lines
xbasc()
imshow(lim + 2*im +1, hotcolormap(4)) // detected lines shown in yellow


// ======= Example 2: how to obtain the parameters
//
// creating a empty picture with a line at y = -90
e = zeros(200,200);
e(10,:) = 1;

// (remember that the Y axis points downwards and is centered in the
// middle of the image)

// getting its hough transform, and finding the points
// corresponding to y=10
[h, rrange] = hough(e);
[r,c] = find(h == max(h))
```

```
// Gets the parameters of the line
theta = c - 1       // 90 degrees
rho   = rrange(r)   // -90 rho (upper half of image)

// thx to Herve Lombaert for inspiring example #2 !

chdir(initial_dir);
```

KNOWN PROBLEMS

There are no parameters for controlling the sampling of the parameter space. Also, the routine is too slow even for medium-sized images. It's purpose is for prototyping. When the function gets better, it will be transcribed to C language.

REFERENCES

"Shape Analysis and Classification", L. da F. Costa and R. M. Cesar Jr., CRC Press.

"Practical Computer Vision using C", J. R. Parker, Wiley.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

ihough, drawline, edge

NAME

ihough − inverse hough transform

SYNOPSIS

im = ihough(ht,dims)

PARAMETERS

im binary image with digital straight lines

ht binarized Hough transform accumulation matrix. Rho varies along rows; theta varies along columns. Theta range is 0-179 degrees w/ 180 samples.

dims vector with the size of the output image. Can be the output of the "size" routine.

DESCRIPTION

Function **ihough** calculates the inverse hough transform of a binary image. It takes as input a binarized hough image, and draws the corresponding digital straight lines in the output image.

EXAMPLE

Cf. help page for hough.

REFERENCES

"Shape Analysis and Classification", L. da F. Costa and R. M. Cesar Jr., CRC Press.

"Digital Image Processing Algorithms", I. Pitas, Prentice Hall.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

hough, drawline, edge

NAME

im2bw − convert images to binary by thresholding

SYNOPSIS

BW = im2bw(Img, level [, maxvalue])
BW = im2bw(Index, map, level [, maxvalue])

PARAMETERS

Img M x N x 3 truecolor image or M x N grayscale intensity image.

index and map
M x N indexed image and its M x 3 colormap.

level the threshold level, between 0 to 1 (a ratio, like percentage). Pixels of the image that are higher or equal than the level are mapped to 1, while pixels that are strictly lower than the level are mapped to zero.

maxvalue
the maximum value to assume Img can take. (Optional, defaults to 1)

DESCRIPTION

Function **im2bw** converts RGB images, indexed images and grayscale intensity images to binary, by thresholding.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

[Img,map] = imread('indian.bmp');
xbasc()
imshow(Img, map);
bw = im2bw(Img, map,0.5);
imshow(bw,2);
bw = im2bw(Img, map,0.2);
imshow(bw,2);

chdir(initial_dir);
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

im2gray

NAME

im2gray − converts color images or colormaps to grayscale

SYNOPSIS

GrayImg = im2gray(RGBImg)
GrayImg = im2gray(index, map)
GrayMap = im2gray(RGBColormap)

PARAMETERS

RGBImg
M x N x 3 truecolor image in any range.

RGBColormap
M x 3 colormap in rgb colorspace.

index and map
M x N indexed image and its M x 3 colormap.

GrayImg
M x N intensity image.

GrayMap
M x 3 colormap.

DESCRIPTION

Function **im2gray** converts RGB images, indexed images and RGB colormaps to grayscale. This is accomplished by converting these objects to YIQ colorspace, make the I and Q channels equal to zero, and finally converting back to RGB colorspace.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

Img = imread('tru.jpg');
Img = im2gray(Img);
xbasc()
imshow(Img);

chdir(initial_dir);
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

im2bw, rgb2ntsc

NAME

imconv − 2D convolution

SYNOPSIS

Outm = imconv(Img, mask [, form])

PARAMETERS

Img M x N Grayscale (intensity) image in any range.

mask n x n matrix, n odd. This is the spatial convolution kernel.

Outm The convolved matrix (grayscale image).

form determines the size of Outm. It assumes be one of the following values:

'same'
Outm has the same size as the original image, M x N. The image is assumed to be 0 outside its bounds.

'full' Outm contains the full convolution, (M+n-1) x (N+n-1). The image is assumed to be 0 outside its bounds.

'valid' Outm contains only the results of the convolution which have been computed within the bounds of *Img*. Outm will be (M-n+1) x (N-n+1).

DESCRIPTION

Function **imconv** performs 2D spatial convolution of a grayscale image and a mask.

edge(Img) Detects edges in *Img*, using the sobel gradient estimator, 0.5 threshold level and in both horizontal and vertical directions.

The other parameters are optional and non-positional. That is, they may be passed to the function by their name. The following example illustrates this.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

// Detect horizontal lines
h = [ -1  -1 -1
       2   2  2
      -1  -1 -1]
img = imread('gra.jpg');
res = imconv(img,h);
imshow(res,[]);
// Detect diagonal lines
d = [ -1  -1  2
      -1   2 -1
       2  -1 -1]
res = imconv(img,d);
imshow(res,[]);

chdir(initial_dir);
```

REMARKS

The kernel is not rotated by 180 degrees. This is in truth a correlation operator, but in practice only symmetric kernels are used.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>, with help from Scilab Group <Scilab@inria.fr>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

mkfilter, edge

NAME

imcorrcoef − template matching by normalized correlation

SYNOPSIS

C = imcorrcoef(Img, template)

PARAMETERS

Img M x N Grayscale image in any range.

template

n x n matrix, n odd. A small object to be found in the image.

C The correlation coefficient image. Its highest value occurs where the template matches exactly.

DESCRIPTION

Function **imcorrcoef** performs template matching by 2D correlation of a grayscale image and a template. This means the template is compared to the image pixel-by-pixel, for every possible translation of the template. The result is normalized to range from -1 to +1. You may also use the absolute value of this measure.

This is an expensive calculation, and should be used only for small templates. The object to be detected must appear in the image with very little variation of rotation and scale. This is useful if you can restrict the image capture to avoid those problems.

EXAMPLE

```
//
// Let's detect the letter "B" in a license plate
//
img = gray_imread(SIPDIR+'images/plate.jpg');
xbasc(),  imshow(img);
template = gray_imread(SIPDIR+'images/template.jpg');
xbasc(),  imshow(template);

C = imcorrcoef(img, template);

xbasc();
imshow(img+2*(C==maxi(C)),[]);  // letter is detected!!
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

imconv, mkfilter, edge

NAME

imfinfo − image file information

SYNOPSIS

info = imfinfo(filename)
info = imfinfo(filename, 'verbose')

PARAMETERS

filename

A string, the image filename to be described. The extension determines the type of the image.

'verbose'

causes *imfinfo* to print image description on the screen.

info A structure (Scilab typed list) containing the descriptions under each of the following fields:

FileName
need no comment.

FileSize
measured in bytes.

Format
"JPEG", "TIFF", "GIF", "BMP", etc.

Width
number of columns.

Height
number of rows.

Depth
bits per pixel.

StorageType
"truecolor" or "indexed".

NumberOfColors
size of the colormap. Equals to zero for truecolor images.

ResolutionUnit
"inch" or "centimeters".

XResolution
number of pixels per *ResolutionUnit* in X direction.

YResolution
number of pixels per *ResolutionUnit* in Y direction.

DESCRIPTION

imfinfo is used to get a description from image files, without reading the pixels into memory. A structure (Scilab typed list) is returned with the information, which can be accessed by their names (e.g. info.Depth, info.FileSize, ...). The 'verbose' option causes the description to be printed in an organised manner.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

info = imfinfo('example.png');     // reads information into a tlist
info.Depth                         // members are accessed by name
info.StorageType
```

```
imfinfo('example.png','verbose');  // pretty-print description

chdir(initial_dir);
```

REMARKS

In theory, **imfinfo** will work with any raster image format. In practice, it has been tested only for BMP, GIF, JPEG, PNG, PCX, TIFF, XPM, and MIFF. **imfinfo** will *probably work* with other formats, but in this case have not been tested by the developers of the SIP toolbox.

BUGS AND SHORTCOMINGS

Images are stored in double precision matrices. Hopefully, the next release will make usage of integer types.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

imwrite, imread, imshow

NAME

imnoise − generate noise (salt & pepper, etc)

SYNOPSIS

J = imnoise(I, type [,parameters])

PARAMETERS

I Input image (grayscale).

J Noisy image (grayscale).

type String having one of these values:
'salt & pepper': drop-out/On-off noise
'speckle': multiplicative noise
'gaussian': Gaussian white/additive noise
'localvar': Pixel-specific variance (Zero-mean Gaussian)

parameters
A sequence of parameters to control the noise distribution, depending on the chosen type. If omitted, default values are used (see below).

DESCRIPTION

imnoise(Img, type) adds a *type* of noise to the intensity image Img. Optionally, you can control the noise parameters starting at the 3rd. argument to imnoise. Here are example of noise types and their params:

J = imnoise(I,'salt & pepper',d) adds drop-out noise, where d is the noise density (probability of swapping a pixel). (default: d=0.05).

J = imnoise(I,'salt & pepper', d, val) does the same, but "val" is the value of salt (defaults to maximum of image). If "val" == 0, then pixels are replaced by uniformly random gray values.

J = imnoise(I,'gaussian',m,v) adds Gaussian additive noise of mean m and variance v. (default: m=0 and v=0.01)

J = imnoise(I,'localvar',V) additive zero-mean Gaussian noise where the variance at Img(i,j) is V(i,j).

J = imnoise(I,'speckle',v) adds multiplicative noise, using J = I+ noise*I, where noise is uniformly distributed with mean 0 and variance v. (default: v=0.04)

The mean and variance parameters are specified as if image intensities went from 0 to 1. By default, we consider that "1" corresponds to the maximum intensity value of the image. If you want to change this for 'gaussian' and 'speckle', pass an extra parameter at the end of the argument list. For instance, your image may have maximum intensity 180 even though the grayscale range is 0-1:

J = imnoise(I,'gaussian', m, v, val)

J = imnoise(I,'speckle', v, val)

EXAMPLE

```
xset('auto clear', 'on');
A = gray_imread(SIPDIR + 'images/gra.jpg');
imshow(A);

N = imnoise(A,'salt & pepper');
imshow(N,[]);
```

```
N = imnoise(A,'salt & pepper',0.3);
imshow(N,[]);


// Replace pixel by independent random value:
N = imnoise(A,'salt & pepper',0.3,0);
imshow(N,[]);

// Replace pixel by *dependent* random value:
N = imnoise(A,'speckle');
imshow(N,[]);

xset('auto clear', 'off');
```

REFERENCES

"Noise Generation", The Hypermedia Image Processing Reference (HIPR), R. Fisher, S. Perkins, A. Walker and E. Wolfart, University of Edinburgh
http://www.dai.ed.ac.uk/HIPR2/noise.htm

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

mogrify '-noise' flag, mkfilter, gsm2d

NAME

imphase − image phase calculation.

SYNOPSIS

phase_value = imphase(formula_name,thresh,im1,im2,im3[,im4,im5])

PARAMETERS

phase_value

a 2D array containing the calculated phase (between (-%pi) and (+%pi)).

formula_name

a string, the name of the formula to use to calculate the phase. May be
'bucket3a' (3 images, phase_shift=(%pi/2)),
'bucket3b' (3 images, phase_shift=(2*%pi/3)),
'bucket3c' (3 images, phase_shift=(2*%pi/3)),
'bucket4a' (4 images, phase_shift=(%pi/2)),
'bucket5a' (5 images, phase_shift=(%pi/2)),
'bucket5b' (5 images, phase_shift=(%pi/2)).

threshold

if the difference for a pixel between 2 images is less than this threshold, than this pixels is considered to be belong to the background.

im1, im2, ...

matrix containing an image.

DESCRIPTION

imphase calculates the phased image resulting from the combination of several phase-shifted pictures.

This function is used in profilometry (or in interferometry) to calculate the phase on any point of an image. The elevation of a point is proportionnal to the calculated of this point.

Here is a usual method for proceeding:
1) Fringes are projected on a reference plane, perpendicular to a CCD camera.
2) A 1st image is taken with the CCD camera.
3) Fringes are deplaced from a quarter of an interfringe (in this case, phase shift = %pi/2)
4) A 2nd image is taken. And so on...
5) The phased image(phref) is calculated from these images. Several algorithms exist depending on
   a) the phase shift (often (2*%pi/3) or (%pi/2))
   b) the number of images available
6) Repeat the same operations with an object instead of the reference plane. Calculate (phobj): the phased image for the object.
7) Unwrap (phref) and (phobj) in (uwphref) and (uwphobj).
8) The altitude map is proportional to (uwphobj)-(uwphref)

Using five images (when it is possible) should lead to better phase calculation.

formula can be: usage: general purpose (vibration environnements) phase=arctan((i3-i2)/(i1-i2))

usage: general purpose (vibration environnements) p1=2*(%pi/3);p2=2*p1; phase=arctan(((i3-i2)+(i1-i3)*cos(p1)+(i2-i1)*cos(p2))/...
((i1-i3)*sin(p1)+(i2-i1)*sin(p2)))

usage: general purpose (vibration environnements) phase=arctan((sqrt(3)*(i3-i2))/(2*i1-i2-i3))

usage: general purpose phase=arctan((i4-i2)/(i1-i3))

usage: phase shift correction (Hariharan) phase=arctan((2*(i4-i2))/(i1-2*i3+i5))

usage: enhanced phase shift error correction (Creath/Schmit) phase=arctan((i1-4*i2+4*i4-i5)/(i1+2*i2-6*i3+2*i4+i5))

Note about Object mask: If the value of a pixel doesn't vary more than the threshold between two images, than we consider that this pixel is in the background (and not on the object on which we project fringes), and we give it the phase = 0 radian.

EXAMPLE

As all this may not be easy for people unused to these
concepts of phase shifting and phase unwrapping,
here is a very detailled (and long) example:
all the steps (1 to 8) above are illustrated.
I found more useful for this example to show how to simulate
several objects

```
//begin
stacksize(4e7);//much memory needed to treat pictures

nb=6;//number of black fringes on the reference plane
step=0.02;//definition of the image
theta=%pi/6;//fringes are projected on the object
//with this angle (compared to the vertical)
//(nb:this script doesn't consider shadows which can occur in real cases)

xmax=5.3;
x=0:step:xmax;
p=(xmax/nb)/cos(theta);//distance between two black fringes
//projected on a plane perpendicular to the camera
//(=interfringe seen from the camera)
y=ones(x);
plan=(x'*y)';//matrix defining the reference plane



//Now we choose an altitude for each point of the plane:
//Below is the simulation of 3 common objects:
//a gaussian, a wedge and a pyramid.
//just uncomment the one you want (here: pyramid)


//Simulation of a gaussian object:
//size_gauss=1.5;//parameter to have a gaussian object large or narrow
//amplitude=5;//maximal height of the object
//relief_line=exp((-(x-2.5).^2)/size_gauss);//these functions can be adapted to
//relief_column=exp((-(x-2.5).^2)/size_gauss);// simulate any object you want
//z=amplitude*relief_column'*relief_line;//compute altitude
```

```
//Simulation of a prism (or a wedge): because of the dicontinuities
//the unwrapping process will make an error in shape reconstitution
//relief_line=ones(x);
//start_point=56;//the prism begins on column 56
//relief_line(start_point:$)=2*(x(start_point:$)-x(start_point))...
//                            +relief_line(start_point-1);
//relief_column=ones(x);
//z=relief_column'*relief_line;//compute altitude
//z(1:54,:)=ones(z(1:54,:));//the prism is only between
//z(100:$,:)=ones(z(100:$,:));//lines 55 to 99


//Simulation of a pyramid
relief_line=ones(x);
start_point=floor(size(x,'c')/2);//the pyramid is centered in the picture
relief_line(1:start_point)=2*(x(1:start_point));
relief_line(start_point:$)=-2*(x(start_point:$)-x(start_point))...
                           +relief_line(start_point-1);
relief_column=ones(x);
relief_column(1:start_point)=2*(x(1:start_point));
relief_column(start_point:$)=-2*(x(start_point:$)-x(start_point))...
                           +relief_column(start_point-1);
//compute altitude:
z=relief_column'*ones(relief_line)+ones(relief_column)'*relief_line;


//phase calculation
phref=2*%pi*(plan/p);//phase of reference (phase of the points of
//the reference plane)
phobj=2*%pi*(plan+z*tan(theta))/p;//phase of the points of the object


//simulate 4 images shifted with (%pi/2) for the reference plane
//and calculate the phased image
phi=phref;
image=zeros(phi);

for a=1:4
dephi=(a-1)*%pi/2;//phase shift=(%pi/2)
lum=127.5*(1+cos(phi+dephi));//calculate luminance of the image
image(:,:,a)=lum;
end

//phase calculation for the reference plane
ph1=imphase('bucket4a',0,image(:,:,1),image(:,:,2),image(:,:,3),image(:,:,4));


//simulate 4 images shifted with (%pi/2) with fringes projected on the object
phi=phobj;
image=zeros(phi);

for a=1:4
dephi=(a-1)*%pi/2;//phase shift=(%pi/2)
lum=127.5*(1+cos(phi+dephi));//calculate luminance of the image
```

```
image(:,:,a)=lum;
end

//phase calculation for the object
ph2=imphase('bucket4a',0,image(:,:,1),image(:,:,2),image(:,:,3),image(:,:,4));



//display the original object on which we projected fringes
xset("window",0);xbasc();
plot3d1(1:8:size(z,'r'),1:8:size(z,'c'),z(1:8:$,1:8:$));
xtitle("original object: relief");

//display one of the images with fringes
xset("window",1);xbasc();imshow(image(:,:,1)/maxi(image(:,:,1));
xtitle(['one of the original images:';'friges are projected on the object']);

//display the phased image of the object
xset("window",2);xbasc();
imshow((ph2+%pi)*(1/(2*%pi)));xtitle('image of phase for the object');



//Now we verify if what we've done is correct:
//we exploit the phased images to find again the profile of the object:
//1st step is phase unwrapping:
disp('phase unwrapping: be a bit patient');
//unwrap the phase corresponding to the referenc plane:
uwphref=unwrapl((ph1+%pi)*(127.5/%pi));
//unwrap the phase corresponding to the object:
uwphobj=unwrapl((ph2+%pi)*(127.5/%pi));


//2nd step:the difference of unwrapped phases is
//proportionnal to the altitude z
uwpht=uwphobj-uwphref;
//3rd step: display
xset("window",3);xbasc();plot3d1(1:size(uwpht,'r'),1:size(uwpht,'c'),uwpht);
xtitle(['phase reconstituted from the wrapped phases';
        'this phase is proportionnal to altitude']);

//end
```

REFERENCES

software FRINGE ANALYSIS by HOLO3 (St Louis, FRANCE)

software INTELLIWAVE by engsynthesis

Optics 505 - James C.Wyant

"Modelisation de forme 3D par methode de moire de projection et analyse par decalage de phases", Cyril Breque & Fabrice Bremand

"Metrologie optique par decalage de phase", Yves Surrel,conservatoire national des arts et metiers

A google search with keywords: phase shifting interferometry or moire or phase unwrapping should lead to good introductory documents on the subject.

AUTHOR

Jocelyn DRUEL <jocelyn.druel1@libertysurf.fr>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://cyvision.if.sc.usp.br/~rfabbri/sip/

SEE ALSO

unwrapl

NAME

improfile − draws intensity profiles of an image

SYNOPSIS

[hprofile,vprofile]=improfile(matrix[,option,maximal_luminance])

PARAMETERS

matrix

The gray-level matrix obtained by matrix=imread("my_picture.jpg").

option

0 to draw the profiles on the image 1 to draw the profiles in separated windows. 2 to draw the profiles on the image and the gaussian interpolation (used with laser beam images) Default=0.

maximal_luminance

hprofile and vprofile will contain values in the 0-1 range. To draw the profile, you can specify a multiplication factor to these values. For example, if your initial image was 8bits, you could pass 255 as the maximal luminanace of the image. Default= 255.

hprofile (or vprofile)

Matrix containing the horizontal (or vertical) profile selected.

DESCRIPTION

Draws the values of pixels on a line and/or a row. Can draw profiles on the image or in separate windows. With the option=2, you can determine radius w of a laser beam, which allows you to find the beam waist.

This function was created (and tested) only with gray level images.

EXAMPLE

```
a=imread(SIPDIR+'images/tru.jpg');
g=im2gray(a);
improfile(g);

//profiles in separate windows
improfile(g,1)

//laser beam visualisation
b=gray_imread(SIPDIR+'images/photonics/laser1.jpg');
improfile(b,2);
```

REMARKS

This function is a bit different than its MatLab equivalent.

AUTHORS

Jocelyn DRUEL <jocelyn.druel1@libertysurf.fr> Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

xgetpixel

NAME

imread − reads image file in ANY format

SYNOPSIS

Image = imread(filename)
[Index, Map] = imread(filename)

PARAMETERS

filename

A string, the image filename to be read. The extension determines the type of the image.

Image For truecolor images, this is a MxNx3 matrix in range 0-1 (double precision); For binary images, *image* is a MxNx1 matrix (0=black and 1=white).

Index MxN matrix containing the indexes to the image colormap (for indexed images). Indices start at 1.

Map Mx3 matrix containing the image colormap. Entries range from 0 to 1 (double precision).

DESCRIPTION

imread reads BMP, GIF, JPEG, PNG, PCX, TIFF, XPM, and even more types of image files into Scilab. The format of the file is inferred from the extension in the *filename* parameter.

Img = imread(filename) reads image in *filename* into *Img* matrix. If *filename* contains a truecolor image, *Img* is a MxNx3 hypermatrix, so for example Img(:,:,1) stands for the red channel. For binary images, Img is a MxNx1 matrix.

[Index,Map] = imread(filename) reads the indexed image in *filename* into *Index* (MxN) and *Map* (Mx3), its colormap. Grayscale and Paletted images are read in this way.

To know if the image stored in *filename* is truecolor or indexed, you can use the **imfinfo** command, and then call **imread** accordingly.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

imfinfo('example.png','verbose');
// the image is indexed
[im,i] = imread('example.png');
xbasc();
imshow(im,i);

chdir(initial_dir);
```

FILE FORMATS

SIP should work with the following formats, but there is no warranty (unless you pay something :-). Please contact us in case of any problems.

Name Mode Description

- 8BIM *rw Photoshop resource format
- AFM *r- TrueType font
- APP1 *rw Photoshop resource format
- ART *r- PF1: 1st Publisher
- AVI *r- Audio/Visual Interleaved
- AVS *rw AVS X image
- BIE *rw Joint Bi-level Image experts Group interchange format
- BMP *rw Microsoft Windows bitmap image
- CAPTION *r+ Caption (requires separate size info)

- CMYK *rw Raw cyan, magenta, yellow, and black samples (8 or 16 bits, depending on the image depth)
- CMYKA *rw Raw cyan, magenta, yellow, black, and matte samples (8 or 16 bits, depending on the image depth)
- CUT *r- DR Halo
- DCM *r- Digital Imaging and Communications in Medicine image
- DCX *rw ZSoft IBM PC multi-page Paintbrush
- DIB *rw Microsoft Windows bitmap image
- DPS *r- Display PostScript
- DPX *r- Digital Moving Picture Exchange
- EPDF *rw Encapsulated Portable Document Format
- EPI *rw Adobe Encapsulated PostScript Interchange format
- EPS *rw Adobe Encapsulated PostScript
- EPS2 *-w Adobe Level II Encapsulated PostScript
- EPS3 *-w Adobe Level III Encapsulated PostScript
- EPSF *rw Adobe Encapsulated PostScript
- EPSI *rw Adobe Encapsulated PostScript Interchange format
- EPT *rw Adobe Encapsulated PostScript with TIFF preview
- FAX *rw Group 3 FAX
- FILE *r- Uniform Resource Locator
- FITS *rw Flexible Image Transport System
- FPX *rw FlashPix Format
- FTP *r- Uniform Resource Locator
- G3 *rw Group 3 FAX
- GIF *rw CompuServe graphics interchange format
- GIF87 *rw CompuServe graphics interchange format (version 87a)
- GRADIENT *r- Gradual passing from one shade to another
- GRANITE *r- Granite texture
- GRAY *rw Raw gray samples (8 or 16 bits, depending on the image depth)
- H *rw Internal format
- HDF -rw Hierarchical Data Format
- HISTOGRAM *-w Histogram of the image
- HTM *-w Hypertext Markup Language and a client-side image map
- HTML *-w Hypertext Markup Language and a client-side image map
- HTTP *r- Uniform Resource Locator
- ICB *rw Truevision Targa image
- ICM *rw ICC Color Profile
- ICO *r- Microsoft icon
- ICON *r- Microsoft icon
- IMPLICIT *--
- IPTC *rw IPTC Newsphoto
- JBG *rw Joint Bi-level Image experts Group interchange format

- JBIG *rw Joint Bi-level Image experts Group interchange format
- JP2 *rw JPEG-2000 JP2 File Format Syntax
- JPC *rw JPEG-2000 Code Stream Syntax
- JPEG *rw Joint Photographic Experts Group JFIF format
- JPG *rw Joint Photographic Experts Group JFIF format
- LABEL *r- Text image format
- LOGO *rw ImageMagick Logo
- M2V *rw MPEG-2 Video Stream
- MAP *rw Colormap intensities (8 or 16 bits, depending on the image depth) and indices (8 or 16 bits, depending on whether colors exceeds 256).
- MAT *-w MATLAB image format
- MATTE *-w MATTE format
- MIFF *rw Magick image format
- MNG *rw Multiple-image Network Graphics
- MONO *rw Bi-level bitmap in least-significant--byte-first order
- MPC -rw Magick Persistent Cache image format
- MPEG *rw MPEG-1 Video Stream
- MPG *rw MPEG-1 Video Stream
- MPR *r- Magick Persistent Registry
- MSL *r- Magick Scripting Language
- MTV *rw MTV Raytracing image format
- MVG *rw Magick Vector Graphics
- NETSCAPE *r- Netscape 216 color cube
- NULL *r- Constant image of uniform color
- OTB *rw On-the-air bitmap
- P7 *rw Xv thumbnail format
- PAL *rw 16bit/pixel interleaved YUV
- PALM *rw Palm Pixmap format
- PBM *rw Portable bitmap format (black and white)
- PCD *rw Photo CD
- PCDS *rw Photo CD
- PCL *-w Page Control Language
- PCT *rw Apple Macintosh QuickDraw/PICT
- PCX *rw ZSoft IBM PC Paintbrush
- PDB *r- Pilot Image Format
- PDF *rw Portable Document Format
- PFA *r- TrueType font
- PFB *r- TrueType font
- PFM *r- TrueType font
- PGM *rw Portable graymap format (gray scale)
- PICON *rw Personal Icon
- PICT *rw Apple Macintosh QuickDraw/PICT
- PIX *r- Alias/Wavefront RLE image format
- PLASMA *r- Plasma fractal image
- PM *rw X Windows system pixmap (color)
- PNG *rw Portable Network Graphics
- PNM *rw Portable anymap
- PPM *rw Portable pixmap format (color)

- PREVIEW *-w Show a preview an image enhancement, effect, or f/x
- PS *rw Adobe PostScript
- PS2 *-w Adobe Level II PostScript
- PS3 *-w Adobe Level III PostScript
- PSD *rw Adobe Photoshop bitmap
- PTIF *rw Pyramid encoded TIFF
- PWP *r- Seattle Film Works
- RAS *rw SUN Rasterfile
- RGB *rw Raw red, green, and blue samples (8 or 16 bits, depending on the image depth)
- RGBA *rw Raw red, green, blue, and matte samples (8 or 16 bits, depending on the image depth)
- RLA *r- Alias/Wavefront image
- RLE *r- Utah Run length encoded image
- ROSE *rw 70x46 Truecolor test image
- SCT *r- Scitex HandShake
- SFW *r- Seattle Film Works
- SGI *rw Irix RGB image
- SHTML *-w Hypertext Markup Language and a client-side image map
- STEGANO *r- Steganographic image
- SUN *rw SUN Rasterfile
- SVG *rw Scalable Vector Gaphics
- TEXT *rw Raw text
- TGA *rw Truevision Targa image
- TIF *rw Tagged Image File Format
- TIFF *rw Tagged Image File Format
- TILE *r- Tile image with a texture
- TIM *r- PSX TIM
- TTF *r- TrueType font
- TXT *rw Raw text
- UIL *-w X-Motif UIL table
- UYVY *rw 16bit/pixel interleaved YUV
- VDA *rw Truevision Targa image
- VICAR *rw VICAR rasterfile format
- VID *rw Visual Image Directory
- VIFF *rw Khoros Visualization image
- VST *rw Truevision Targa image
- WBMP *rw Wireless Bitmap (level 0) image
- WMF *r- Windows Metafile
- WPG *r- Word Perfect Graphics
- X *rw X Image
- XBM *rw X Windows system bitmap (black and white)
- XC *r- Constant image uniform color
- XCF *r- GIMP image
- XML *r- Scalable Vector Gaphics
- XPM *rw X Windows system pixmap (color)
- XV *rw Khoros Visualization image
- XWD *rw X Windows system window dump (color)
- YUV *rw CCIR 601 4:1:1

Modes:

* Native support (no call to external utility)
r Read
w Write

REMARKS

The new **imshow** is incompatible with the same function from sip-0.2.0 and previous versions. Its behavior has been made more similar to that of comercial packages similar to Scilab.

BUGS AND SHORTCOMINGS

Images are stored in double precision matrices. Hopefully, the next release will also make use of integer types.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

imwrite, gray_imread, im2gray, imfinfo, imshow

NAME

imroi − roi (region of interest) of an image

SYNOPSIS

mask=imroi(image[,type_of_mask,option])

PARAMETERS

image A gray, color or binary image.

type_of_mask

´rect': rectangular or ´ellipse' are the only available selections by now.

option

0 displays no coordinates (default)
1 displays coordinates

mask Binary matrix (0 and 1) corresponding to the image selection. It has the same size as the input image (e.g. it is 3D if the input is 3D, and 2D if the input is 2D).

DESCRIPTION

The function allows the user to select a Region of Interest (ROI) in an image ALREADY DISPLAYED, and returns the corresponding binary matrix.

The mask is working with any of binary, gray or color images.

With the ellipse selection, you draw the rectangle in which the ellipse is drawn.

EXAMPLE

```
a=imread(SIPDIR+'images/ararauna.jpg');
xset("window",0);xbasc();
imshow(a);
mask=imroi(a);
//the user must then select the ROI with the mouse
b=a.*mask;
xset("window",1);xbasc();
imshow(b);//displays just the ROI
```

REMARKS

The Matlab equivalent is roipoly, whose interface is a bit more complicated.

With Scilab-2.6, for color images, the image.*mask is not working (problem of matrix sizes). One possibility to solve this:
image(1,1,1)=image(1,1,1);
then you can do
image.*mask; With Scilab-2.7 there is no problem.

TIP

To generate much more complex ROI's, you can use the Gimp (GNU Image Manipulation Program) <www.gimp.org>, although of course this is not an integrated solution.

AUTHORS

Jocelyn DRUEL <jocelyn.druel1@libertysurf.fr>

The code to display coordinates comes directly from the function xgetpixel by Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

imread, xgetpixel, GIMP

NAME

imshow − displays images in scilab graphic window

SYNOPSIS

imshow(Img)
imshow(Img,n)
imshow(Img,[low high])
imshow(Img,Map)
imshow(RGBImg)
imshow filename
imshow(Img, arg2 [, strf])

PARAMETERS

filename
A String, the image filename to be displayed. The extension determines the type of the image.

Img MxN matrix, a grayscale (0-1 range) or pseudo-color/indexed (1-Ncolors range) image.

n A scalar, the number of levels of gray to display *Img*.

[low high]
1x2 array corresponding to the grayscale range to be considered.

Map Mx3 matrix containing the image colormap (for indexed images). Entries may range from 0 to 1

RGBImg
MxNx3 hypermatrix in 0-1 range.

strf this is used to e.g. put axes around your image. See the plot2d help page. You may use this optional parameter in the assignment style, as in:
imshow(img, strf='041') //same as imshow but draw axes

DESCRIPTION

imshow(Img,n) displays *Img* using *n* gray levels. *Img* is an MxN matrix in 0 - (n-1) range. If *n* is omitted, it will be assumed to be 256.

imshow(Img,[low high]) displays *Img* using a grayscale within the specified range. Elements <= *low* will map onto black, Elements >= *high* will map onto white, and elements in between will map as a shade of gray. If an empty matrix '[]' is used, [low,high] will be assumed to be [mini(Img), maxi(Img)].

imshow(Img,Map) displays *Img* using the specified colormap.

imshow(RGBImg) displays the *RGBImg* MxNx3 (0-1) truecolor image.

imshow(filename) displays the image (*filename*) in scilab graphic. Any raster image format is accepted (see REMARKS section).

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

a = rand(100,100);  // create random image
xbasc();
imshow(a);

imshow(a*255 + 1,hotcolormap(256));
imshow(a,[0.3 0.6]);
imshow(a,[]);
xbasc();
imshow('example.png');
```

chdir(initial_dir);

REMARKS

In theory, **imshow(filename)** will work with any raster image format. In practice, it has been tested only for BMP, GIF, JPEG, PNG, PCX, TIFF, XPM, and MIFF. Displaying other file formats will *probably work*, but this case have not been tested by the developers of the SIP toolbox.

BUGS AND SHORTCOMINGS

The display of truecolor images is somewhat inefficient.

Images are stored in double precision matrices. Hopefully, the next release will make usage of integer types.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

imread, imwrite, imfinfo, xgetpixel

NAME

imvariance − calculates the variance of an image

SYNOPSIS

variance_matrix = imvariance(image)

PARAMETERS

image A gray-level image.

DESCRIPTION

This function computes a matrix containing the variance of each point of an image.

The variance is the sum of the absolute value of the differences between the central pixel and its neighbours:

variance=sum(|x(neighbour)-x(central)|)

A low variance value means a pixel is not very different from its neigbours (in all directions).

This property can be used to unwrap phased images. In case of a "path-following algorithm", the variance can be a "merit function" used to determine which pixels should be connected first. This "merit function" is much more noise immune than a "merit function" based on a laplace kernel.

This algorithm calculates the variance everywhere even on the edges. In some cases, consider multiplying by a mask like this
[8/3 8/5 8/5...;
8/5 1 1...;
so that edge values are really significatives.

EXAMPLE

```
stacksize(1e7); // images are very much memory consumming...

varian=imvariance(imread(SIPDIR+'images/photonics/pyramide_wrapped.jpg'));

imshow(varian/max(varian));// high levels (blank on the image)
                           //represent points where intensity
                           //changes quickly
```

AUTHORS

Jocelyn DRUEL <jocelyn.druel1@libertysurf.fr>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

unwrapl, mkfilter('laplace1')

NAME

imwrite − writes to an image file in ANY format

SYNOPSIS

imwrite(Image, filename, [opt_args])
imwrite(Index, Map, filename, [opt_args])

PARAMETERS

filename

a string, the image filename to be created. The extension determines the type of the image.

Image For truecolor images, this is a MxNx3 hypermatrix in 0-1 range; For binary images or intensity grayscale images, it is a MxNx1 matrix in 0-1 range.

Index MxN matrix containing the indexes to the image colormap (for indexed images). Indices start at 1.

Map Mx3 matrix containing the image colormap (for indexed images). Entries range from 0 to 1

[opt_args]

This is a sequence of statements key1=value1, key2=value2,... where key1, key2,... can be one of the following:

quality

sets the quality level for some formats of images that accept this parameter, like JPEG. Ranges from 1 to 100 (default 75).

In the future, other options may be added, such as dithering and compression type.

DESCRIPTION

imwrite creates BMP, GIF, JPEG, PNG, PCX, TIFF, XPM, and even more types of image files from Scilab elements. The format of the file is inferred from the extension in the *filename* parameter.

imwrite(Img, filename) creates an image *filename* in disk, from *Img* matrix. For truecolor images, *Img* is a MxNx3 hypermatrix, so for example Img(:,:,1) stands for the red channel. For binary images or intensity grayscale images, Img is a MxNx1 matrix.

imwrite(Index, Map, filename) does the same work, but from *Index* (MxN) and *Map* (Mx3) matrices. Grayscale and Paletted images are written in this way.

EXAMPLE

```
a=rand(100,150);    // create a random image
xbasc();
imshow(a);
imwrite(a,'SIPtmp.jpg');
xbasc();
imshow('SIPtmp.jpg');
// now try viewing this image with an ordinary viewer to see
// how it really worked!
```

FILE FORMATS

SIP should work with the following formats, but there is no waranty (unless you pay something :-). Please contact us in case of any problems.

Name Mode Description

- 8BIM *rw- Photoshop resource format
- AFM *r-- TrueType font
- APP1 *rw- Photoshop resource format
- ART *r-- PF1: 1st Publisher
- AVI *r-- Audio/Visual Interleaved

- o AVS *rw+ AVS X image
- o BIE *rw- Joint Bi-level Image experts Group interchange format
- o BMP *rw+ Microsoft Windows bitmap image
- o CAPTION *r+ Caption (requires separate size info)
- o CMYK *rw- Raw cyan, magenta, yellow, and black samples (8 or 16 bits, depending on the image depth)
- o CMYKA *rw- Raw cyan, magenta, yellow, black, and matte samples (8 or 16 bits, depending on the image depth)
- o CUT *r-- DR Halo
- o DCM *r-- Digital Imaging and Communications in Medicine image
- o DCX *rw+ ZSoft IBM PC multi-page Paintbrush
- o DIB *rw+ Microsoft Windows bitmap image
- o DPS *r-- Display PostScript
- o DPX *r-- Digital Moving Picture Exchange
- o EPDF *rw- Encapsulated Portable Document Format
- o EPI *rw- Adobe Encapsulated PostScript Interchange format
- o EPS *rw- Adobe Encapsulated PostScript
- o EPS2 *-w- Adobe Level II Encapsulated PostScript
- o EPS3 *-w- Adobe Level III Encapsulated PostScript
- o EPSF *rw- Adobe Encapsulated PostScript
- o EPSI *rw- Adobe Encapsulated PostScript Interchange format
- o EPT *rw- Adobe Encapsulated PostScript with TIFF preview
- o FAX *rw+ Group 3 FAX
- o FILE *r-- Uniform Resource Locator
- o FITS *rw- Flexible Image Transport System
- o FPX *rw- FlashPix Format
- o FTP *r-- Uniform Resource Locator
- o G3 *rw- Group 3 FAX
- o GIF *rw+ CompuServe graphics interchange format
- o GIF87 *rw- CompuServe graphics interchange format (version 87a)
- o GRADIENT *r-- Gradual passing from one shade to another
- o GRANITE *r-- Granite texture
- o GRAY *rw+ Raw gray samples (8 or 16 bits, depending on the image depth)
- o H *rw- Internal format
- o HDF -rw+ Hierarchical Data Format
- o HISTOGRAM *-w- Histogram of the image
- o HTM *-w- Hypertext Markup Language and a client-side image map
- o HTML *-w- Hypertext Markup Language and a client-side image map
- o HTTP *r-- Uniform Resource Locator
- o ICB *rw+ Truevision Targa image
- o ICM *rw- ICC Color Profile
- o ICO *r-- Microsoft icon

- ICON *r-- Microsoft icon
- IMPLICIT *---
- IPTC *rw- IPTC Newsphoto
- JBG *rw+ Joint Bi-level Image experts Group interchange format
- JBIG *rw+ Joint Bi-level Image experts Group interchange format
- JP2 *rw- JPEG-2000 JP2 File Format Syntax
- JPC *rw- JPEG-2000 Code Stream Syntax
- JPEG *rw- Joint Photographic Experts Group JFIF format
- JPG *rw- Joint Photographic Experts Group JFIF format
- LABEL *r-- Text image format
- LOGO *rw- ImageMagick Logo
- M2V *rw+ MPEG-2 Video Stream
- MAP *rw- Colormap intensities (8 or 16 bits, depending on the image depth) and indices (8 or 16 bits, depending on whether colors exceeds 256).
- MAT *-w+ MATLAB image format
- MATTE *-w+ MATTE format
- MIFF *rw+ Magick image format
- MNG *rw+ Multiple-image Network Graphics
- MONO *rw- Bi-level bitmap in least-significant-byte-first order
- MPC -rw- Magick Persistent Cache image format
- MPEG *rw+ MPEG-1 Video Stream
- MPG *rw+ MPEG-1 Video Stream
- MPR *r-- Magick Persistent Registry
- MSL *r-- Magick Scripting Language
- MTV *rw+ MTV Raytracing image format
- MVG *rw- Magick Vector Graphics
- NETSCAPE *r-- Netscape 216 color cube
- NULL *r-- Constant image of uniform color
- OTB *rw- On-the-air bitmap
- P7 *rw+ Xv thumbnail format
- PAL *rw- 16bit/pixel interleaved YUV
- PALM *rw- Palm Pixmap format
- PBM *rw+ Portable bitmap format (black and white)
- PCD *rw- Photo CD
- PCDS *rw- Photo CD
- PCL *-w- Page Control Language
- PCT *rw- Apple Macintosh QuickDraw/PICT
- PCX *rw- ZSoft IBM PC Paintbrush
- PDB *r-- Pilot Image Format
- PDF *rw+ Portable Document Format
- PFA *r-- TrueType font
- PFB *r-- TrueType font
- PFM *r-- TrueType font
- PGM *rw+ Portable graymap format (gray scale)
- PICON *rw- Personal Icon
- PICT *rw- Apple Macintosh QuickDraw/PICT
- PIX *r-- Alias/Wavefront RLE image format

- PLASMA *r-- Plasma fractal image
- PM *rw- X Windows system pixmap (color)
- PNG *rw- Portable Network Graphics
- PNM *rw+ Portable anymap
- PPM *rw+ Portable pixmap format (color)
- PREVIEW *-w- Show a preview an image enhancement, effect, or f/x
- PS *rw+ Adobe PostScript
- PS2 *-w+ Adobe Level II PostScript
- PS3 *-w+ Adobe Level III PostScript
- PSD *rw- Adobe Photoshop bitmap
- PTIF *rw- Pyramid encoded TIFF
- PWP *r-- Seattle Film Works
- RAS *rw+ SUN Rasterfile
- RGB *rw+ Raw red, green, and blue samples (8 or 16 bits, depending on the image depth)
- RGBA *rw+ Raw red, green, blue, and matte samples (8 or 16 bits, depending on the image depth)
- RLA *r-- Alias/Wavefront image
- RLE *r-- Utah Run length encoded image
- ROSE *rw- 70x46 Truecolor test image
- SCT *r-- Scitex HandShake
- SFW *r-- Seattle Film Works
- SGI *rw+ Irix RGB image
- SHTML *-w- Hypertext Markup Language and a client-side image map
- STEGANO *r-- Steganographic image
- SUN *rw+ SUN Rasterfile
- SVG *rw+ Scalable Vector Gaphics
- TEXT *rw+ Raw text
- TGA *rw+ Truevision Targa image
- TIF *rw+ Tagged Image File Format
- TIFF *rw+ Tagged Image File Format
- TILE *r-- Tile image with a texture
- TIM *r-- PSX TIM
- TTF *r-- TrueType font
- TXT *rw+ Raw text
- UIL *-w- X-Motif UIL table
- UYVY *rw- 16bit/pixel interleaved YUV
- VDA *rw+ Truevision Targa image
- VICAR *rw- VICAR rasterfile format
- VID *rw+ Visual Image Directory
- VIFF *rw+ Khoros Visualization image
- VST *rw+ Truevision Targa image
- WBMP *rw- Wireless Bitmap (level 0) image
- WMF *r-- Windows Metafile
- WPG *r-- Word Perfect Graphics
- X *rw- X Image
- XBM *rw- X Windows system bitmap (black and white)
- XC *r-- Constant image uniform color
- XCF *r-- GIMP image
- XML *r-- Scalable Vector Gaphics

- o XPM *rw- X Windows system pixmap (color)
- o XV *rw+ Khoros Visualization image
- o XWD *rw- X Windows system window dump (color)
- o YUV *rw- CCIR 601 4:1:1

Modes:

- * Native blob support (no call to external utility)
- r Read
- w Write
- + Multi-image

REMARKS

In writing a binary image, there is NO MORE NEED to multiply it by 65535.

The file created by **imwrite** is not necessarily indexed or truecolor. Commands of the form imwrite(I,M,filename) have a greater chance of creating an indexed image, and imwrite(RGB,filename) have a good chance of creating a truecolor image. However, the final result depends on the characteristics of the image format. For example, JPEG images are always truecolor or grayscale. Typing imwrite(Index,Map,'foo.jpg') will necessarily create a truecolor image. But imwrite(Index,Map,'foo.gif') will certainly create an indexed image.

BUGS AND SHORTCOMINGS

Images are stored in double precision matrices. Hopefully, the next release will make use of integer types.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

imread, imfinfo, imshow

NAME

ind2rgb − convert indexed images to truecolor storage

SYNOPSIS

RGB = ind2rgb(Index, map)

PARAMETERS

index and map

M x N indexed image and its M x 3 colormap.

RGB M x N x 3 truecolor image in the 0-1 range.

DESCRIPTION

Function **ind2rgb** converts indexed image storage to RGB (direct) truecolor storage.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

[Img,map] = imread('indian.bmp');
xbasc()
imshow(Img, map);
RGB = ind2rgb(Img, map);
imshow(RGB);

chdir(initial_dir);
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

im2gray, im2bw

NAME

interferometry_gui − Graphical User Interface (GUI) for SIP functions

DESCRIPTION

This is a quick start guide: first of all, this GUI was written to make tests in the photonic field, which means you'll find many functions related to laser images.

As it is highly customizable, you can adapt it easily to your needs.

Here is a description of a few experiments and pictures:
1) laser1.jpg is a picture of a laser beam magnified by an microscope objective X10 and filtered by a pinhole of a few microns (=spatial filter).
Operation > Profiles show the intensity profiles.
LaserBeam > Find Gaussian Profiles allows modelling of these profiles by a gaussian curve (laser beam should have a gaussian profile). The beam waist can be deducted from these values.
The laser speckle can be smoothed by one (or several) median filtering(s) found in Operations: this filter removes the high-frequency noise.

2) speckle1.png and speckle2.png are 2 images of an experiment in speckle interferometry: a Michelson interferometer is created but the 1st mirror is replaced by a rugged metallic piece and the second one is replaced by a metallic rule.
A CCD Camera saves a first image.
The rule is bent. The Camera take a second picture.
Try, Open > speckle1.png then Operations > substract 2 images (absolute value).
Fringes appear: between 2 dark fringes, the rule has moved from a distance equal to the light wavelength/2 (here, 633/2=316.5 nm).
Normalization, filtering can help better visualization of the fringes.

3) pyramide_wrapped.jpg is what is called a phased image: it was obtained by projecting fringes on an object then on a reference plane.
The goal is to modelize the object in 3D. Operations > Profiles show that luminance is somewhat proportional to altitude, but each time its value reach 255, there's a jump and the luminance restart from zero.
The phase unwrapping process is intended to remove these jumps.
PhasedImages > Unwrap linearly can do it with a very good quality image (like pyramide_wrapped.jpg).
A more complex algorithm is used by Unwrap by path following: it can unwrap more difficult images. You can test with the merit function called Variance: it's usually the best choice.
Be a bit patient when launching those functions: it takes some time (about 2 min on a recent computer).

AUTHORS

Jocelyn DRUEL <jocelyn.druel1@libertysurf.fr>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

unwrapp, unwrapl, improfile

NAME

minmax − Min/Max algorithm for noise removal from images

SYNOPSIS

Output = minmax(Input, [NSteps, StepSize, Adapt, NonAdaptThreshold,
IntMaskSize, ExtMaskSize])

INPUT PARAMETERS

Input a matrix containing a gray-scale image to be filtered by min/max algorithm.

NSteps

number of steps to perform. Default is 10.

StepSize

the step increment for the iterative procedure. Default is 0.05.

Adapt

indicates if the algorithm should adapt itself to the local image gray level or if it considers the NonAdaptThreshold value for defining light and dark regions. Default is FALSE.

NonAdaptThreshold

If Adapt is FALSE, intensity values greater than NonAdaptThreshold will be considered as light regions.

IntMaskSize

Size of the Internal window in which curvature values will be taken into account for deciding Min or Max curvature flow. Default is 1.

ExtMaskSize

Size of the External window in which curvature values will be taken into account for deciding Min or Max curvature flow in the *Adapt mode. Default is 0.*

OUTPUT PARAMETERS

Output

a matrix containing the filtered image.

DESCRIPTION

Function **minmax** filters a gray-scale image using curvature-guided surface evolution. Object borders remain sharp while low-scale noise is removed.

EXAMPLE

```
M = gray_imread(SIPDIR+'images/noisypoly.bmp');
subplot(1,2,1);
imshow(M);
new_M = minmax(M, NSteps=30);
subplot(1,2,2);
imshow(new_M);
```

REFERENCES

"Image Processing via Level Set Curvature Flow ", Malladi, R., and Sethian, J.A., Proceedings of the National Academy of Sciences, Vol. 92(15), pp. 7046-7050, July 1995
http://math.berkeley.edu/~sethian/Movies/Movienoiseremoval.html

AUTHORS

Leandro Estrozi (lfestrozi@if.sc.usp.br)

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

curvature, curvature2d, mogrify

NAME

mkfftfilter − builds 2D frequency-domain filters

SYNOPSIS

transfer_function = mkfftfilter(image,filtername,frequency1[,frequency2])

INPUT PARAMETERS

image A gray-level image.

filtername

a string, the name can be **'binary'**, **'butterworth1'**, **'butterworth2'**,

frequency1, frequency2

1st and 2nd cut-off frequencies which set the filter characteristics.

OUTPUT PARAMETERS

transfer_function

is a matrix with values between 0 and 1. These values can then be applied on the fft spectrum of an image.

DESCRIPTION

This function gives some popular filters to be applied on the spectrum (fft) of an image.

The Fourier Transform gives informations about which frequencies are present in a signal (=spectrum).

A great property of the spectrum is that the original image can be reconstructed from it. Of course, modifications in the spectrum will result in a modified image, but spectrum modifications can be easier and more intuitive.

A combination of several filters is possible.

All these filters are cylindrical and act only on amplitude (not on phase). The following filters are available (h is the trasfer function):

h=1/(1+(f/frequency1)ˆ(2*n)) n=1,2 or 3 for 'butterworth1', 'butterworth2' or 'butterworth3'.

'exp': The exponential filter: h=exp(-(f/frequency1)ˆ1);

The gaussian filter which is a particular case of the exponential: h=exp(-(f/frequency1)ˆ2);

'trapeze': h=1 if f<=frequency1 h=-(f-frequency2)/(frequency2-frequency1) h=0 if f>=frequency2

EXAMPLE

```
stacksize(4e7); // increase the stack size because
                // images are very much memory consumming

image=gray_imread(SIPDIR+'images/ararauna.png');
xset("window",0);xbasc();imshow(image);
xtitle("Original Image");

IM=fft(image,-1);
```

```
//calculate the power spectrum
spectrum=real((IM).*conj(IM));
//for visualisation: the low frequencies are moved to the center of the image
//with sip_fftshift,
//the use of log(spectrum+1) allows to observe great amplitude variations.
xset("window",1);xbasc();imshow(sip_fftshift(log(spectrum+1)),[]);
xtitle("Power Spectrum");

//transfer function
h=mkfftfilter(image,'binary',20);
xset("window",2);xbasc();imshow(h);
xtitle("Transfer Function");

IM2=IM.*sip_fftshift(h);//spectrum modification

//reverse transform
im2=real(fft(IM2,1));
xset("window",3);xbasc();xselect();imshow(im2,[]);
xtitle("Low-pass binary filtering");

//High-pass filter
IM3=IM.*sip_fftshift(1-h);//spectrum modification with (1-h)
im3=real(fft(IM3,1));
xset("window",4);xbasc();xselect();imshow(im3,[]);
xtitle("High-pass binary filtering");

//Combination of 2 filters
h1=mkfftfilter(image,'binary',30);
h2=mkfftfilter(image,'binary',5);
h=h1-h2;
IM4=IM.*sip_fftshift(h);//spectrum modification
im4=real(fft(IM4,1));
xset("window",5);xbasc();xselect();imshow(im4,[]);
xtitle("Band-pass binary filtering");
```

AUTHOR

Jocelyn DRUEL <jocelyn.druel1@libertysurf.fr>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

fft, gsm2d, imconv

NAME

mkfilter − returns popular 2D convolution kernels

SYNOPSIS

K = mkfilter(name)

PARAMETERS

name a string, the name of the filter mask. May be 'sobel', 'prewitt', 'laplace1', ´laplace2', 'laplace3','sh1' (or 'sharp1'), 'sh2' (or 'sharp2'), 'low-pass', ´mean', 'circular', 'circular-mean'. In the future there will be more options.

K a 2D array containing the convolution kernel.

DESCRIPTION

mkfilter builds well-known 2D filter "masks" (kernels), such as sobel, prewitt, mean, etc. to be used together with a function such as *imconv*.

K = mkfilter('sobel') returns a 3x3 edge-finding and y-derivative approximation filter. To find vertical edges, use -K'.

K = mkfilter('prewitt') returns another 3x3 edge-finding and y-derivative approximation filter. To find vertical edges, use -K'.

K = mkfilter('laplace1') returns a 3x3 kernel which shows points of an image where intensity is varying quickly. The "laplacien" of an image is the two-dimensionnal second derivative. Because images are discrete (and not continuous), the "laplacien" can only be approximated. The 3 different kernels often used to estimate it are given by "laplace1", "laplace2" and "laplace3". These kernels can be used to detect edges of an image.

K = mkfilter('sh1'): "sharp enhancer". Returns a 3x3 kernel which renforce high frequencies of the image: it accentuates the variations of a pixel compared to its neighbours. Problem: it enhances the noise too (it may be useful to denoise the image before).

K = mkfilter('sh2') has the same effect than "sh1" but its coefficients are more powerfull.

K = mkfilter('low-pass') : this is a low-pass filter. The goal is inverse of sharp enhancer filters - the image is smoothed. This kernel is only one of the possible kernels.

K = mkfilter('mean') : this is another low-pass filter. The mean value of the central pixel and its neighbours is affected at the central pixel.

K = mkfilter('circular',rad) is an euclidean circular mask with radius "rad" pixels, whose elements are all 1.

K = mkfilter('circular-mean',rad) is a low pass filter, the same as 'circular', but the matrix K is divided by the number of 1-elements.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

Img = imread('tru.jpg');
Img = im2gray(Img);
xbasc()
imshow(Img);
sob = mkfilter('sobel');  // stamp effect
```

```
A = imconv(Img, sob);
xbasc()
imshow(A, [])

chdir(initial_dir);
```

REMARKS

The Matlab equivalent to mkfilter is "fspecial".

REFERENCES

"Processamento Digital de Imagens", O. M. Filho and H. V. Neto, Brasport -- Rio de Janeiro, pp. 47-56.

"Algorithms for Image Processing and Computer Vision", J.R. Parker, Wiley, chapter 1.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>
Jocelyn Druel <jocelyn.druel1@libertysurf.fr>
Leandro Estrozi <lfestrozi@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

imconv, edge

NAME

mogrify − filter, rotate, zoom, equalization, and MANY more.

SYNOPSIS

[imout (,map)] = mogrify(im, argv);
[imout (,map)] = mogrify(imRGB, argv);
[imout (,map)] = mogrify(index, colormap, argv);

INPUT PARAMETERS

im binary or grayscale image in the 0-1 range

imRGB
truecolor image represented as a 3D array, in the 0-1 range.

index a 2D array in the 1-NC range, where NC is the number of colors.

colormap
an (NCx3) matrix in the 0-1 range.

argv (argument vector), a string vector, row or column, containing image processing commands. See the section "ARGV ARGUMENT" bellow for a listing of the available options. The "EXAMPLE" bellow demonstrates how to use this argument.

OUTPUT PARAMETERS

imout For truecolor images, this is a MxNx3 matrix in the 0-1 range. For binary images, *image* is a MxNx1 matrix (0=black and 1=white). For grayscale images, this is an MxN matrix in the 0-1 range. For indexed images, this is an MxN matrix in the 1-NC range, where NC is the number of colors.

map NCx3 matrix containing the image colormap, qhere Nc is the number of colors. Entries range from 0 to 1.

DESCRIPTION

Function **mogrify** does many image processing operations. It is a direct C-language interface to ImageMagick, providing the same functinality as the command-line "mogrify" utility. The difference is that SIP's **mogrify** acts upon scilab matrices instead of image files, and, therefore, is considerably faster. Please see the section "ARGV ARGUMENT" bellow for details on how the parameter *argv* is built.

EXAMPLE

Please follow the TransMogrify demo:

```
exec(SIPDEMO);
```

NEWS

mogrify is now working with truecolor and paletted images.

ARGV ARGUMENT

In this section, angle brackets ("<>") enclose variables and curly brackets ("{}") enclose optional parameters. For example, "**-fuzz** <**distance**>{%}" means you can use the option ['-fuzz','10'] or ['-fuzz', '2%'].

-affine <*matrix*>
drawing transform matrix

This option provides a transform matrix {sx,rx,ry,sy,tx,ty} for use by subsequent **-draw** or **-transform** option.

-antialias
remove pixel aliasing

By default antialiasing algorithms are used when drawing objects (e.g. lines) or rendering vector formats (e.g. WMF and Postscript). Use +antialias to disable use of antialiasing algorithms. Reasons to disable antialiasing include avoiding increasing colors in the image, or improving rendering speed.

-background <*color*>
the background color

The color is specified using the format described in the "Color Names" section of *X(1)*.

-blur <*radius*>x<*sigma*>
blur the image with a Gaussian operator

Blur with the given radius and standard deviation (sigma). It is better if the radius be greater than sigma, but the larger it is, the slower is the processing. If a radius equal to 0 is passed, then mogrify uses the optimal radius.

-border <*width*>x<*height*>
surround the image with a border of color

See **-geometry** for details about the geometry specification.

-bordercolor <*color*>
the border color

The color is specified using the format described in the "Color Names" section of *X(1)*.

-channel <*type*>
the type of channel

Choose from: **Red**, **Green**, **Blue**, **Cyan**, **Magenta**, **Yellow**, or **Black**.

Use this option to extract a particular *channel* from the image.

-charcoal <*factor*>
simulate a charcoal drawing

-colorize <*value*>
colorize the image with the pen color

Specify the amount of colorization as a percentage. You can apply separate colorization values to the red, green, and blue channels of the image with a colorization value list delineated with slashes (e.g. 0/0/50).

-colorspace <*value*>
the type of colorspace

Choices are: **GRAY**, **OHTA**, **RGB**, **Transparent**, **XYZ**, **YCbCr**, **YIQ**, **YPbPr**, **YUV**, or **CMYK**.

Color reduction, by default, takes place in the RGB color space. Empirical evidence suggests that distances in color spaces such as YUV or YIQ correspond to perceptual color differences more closely than do distances in RGB space. These color spaces may give better results when color reducing an image. Refer to quantize for more details.

The **Transparent** color space behaves uniquely in that it preserves the matte channel of

the image if it exists.

The **-colors** or **-monochrome** option is required for this option to take effect.

-contrast
enhance or reduce the image contrast

This option enhances the intensity differences between the lighter and darker elements of the image. Use **-contrast** to enhance the image or **+contrast** to reduce the image contrast.

-crop <*width*>x<*height*>{+-}<*x*>{+-}<*y*>{%}
preferred size and location of the cropped image

See **-geometry** for details about the geometry specification.

The width and height give the size of the image that remains after cropping, and x and y are offsets that give the location of the top left corner of the cropped image with respect to the original image. To specify the amount to be removed, use **-shave** instead.

To specify a percentage width or height to be removed instead, append **%**. For example to crop the image by ten percent (five percent on each side of the image), use **-crop 10%**.

If the x and y offsets are present, a single image is generated, consisting of the pixels from the cropping region. The offsets specify the location of the upper left corner of the cropping region measured downward and rightward with respect to the upper left corner of the image. If the **-gravity** option is present with *NorthEast, East,* or *SouthEast* gravity, it gives the distance leftward from the right edge of the image to the right edge of the cropping region. Similarly, if the **-gravity** option is present with *SouthWest, South,* or *SouthEast* gravity, the distance is measured upward between the bottom edges.

If the x and y offsets are omitted, a set of tiles of the specified geometry, covering the entire input image, is generated. The rightmost tiles and the bottom tiles are smaller if the specified geometry extends beyond the dimensions of the input image.

-cycle <*amount*>
displace image colormap by amount

Amount defines the number of positions each colormap entry is shifted.

-debug
enable debug printout

-despeckle
reduce the speckles within an image

-dither
apply Floyd/Steinberg error diffusion to the image

The basic strategy of dithering is to trade intensity resolution for spatial resolution by averaging the intensities of several neighboring pixels. Images which suffer from severe contouring when reducing colors can be improved with this option.

The **-colors** or **-monochrome** option is required for this option to take effect.

Use **+dither** to turn off dithering and to render PostScript without text or graphic aliasing.

-draw *<string>*

annotate an image with one or more graphic primitives

Use this option to annotate an image with one or more graphic primitives. The primitives include shapes, text, transformations, and pixel operations. The shape primitives are

point	x,y
line	x0,y0 x1,y1
rectangle	x0,y0 x1,y1
roundRectangle	x0,y0 x1,y1 wc,hc
arc	x0,y0 x1,y1 a0,a1
ellipse	x0,y0 rx,ry a0,a1
circle	x0,y0 x1,y1
polyline	x0,y0 ... xn,yn
polygon	x0,y0 ... xn,yn
Bezier	x0,y0 ... xn,yn
path	path specification
image	operator x0,y0 w,h filename

The text primitive is

text	x0,y0 string

The transformation primitives are

rotate	degrees
translate	dx,dy
scale	sx,sy
skewX	degrees
skewY	degrees

The pixel operation primitives are

color	x0,y0 method

The shape primitives are drawn in the color specified in the preceding **-stroke** option. Except for the **line** and **point** primitives, they are filled with the color specified in the preceding **-fill** option. For unfilled shapes, use -fill none.

Point requires a single coordinate.

Line requires a start and end coordinate.

Rectangle expects an upper left and lower right coordinate.

RoundRectangle has the upper left and lower right coordinates and the width and height of the corners.

Circle has a center coordinate and a coordinate for the outer edge.

Use **Arc** to circumscribe an arc within a rectangle. Arcs require a start and end point as well as the degree of rotation (e.g. 130,30 200,100 45,90).

Use **Ellipse** to draw a partial ellipse centered at the given point with the x-axis and y-

axis radius and start and end of arc in degrees (e.g. 100,100 100,150 0,360).

Finally, **polyline** and **polygon** require three or more coordinates to define its boundaries. Coordinates are integers separated by an optional comma. For example, to define a circle centered at 100,100 that extends to 150,150 use:

['-draw'; 'circle 100,100 150,150']

Paths (See Paths) represent an outline of an object which is defined in terms of moveto (set a new current point), lineto (draw a straight line), curveto (draw a curve using a cubic Bezier), arc (elliptical or circular arc) and closepath (close the current shape by drawing a line to the last moveto) elements. Compound paths (i.e., a path with subpaths, each consisting of a single moveto followed by one or more line or curve operations) are possible to allow effects such as "donut holes" in objects.

Use **image** to composite an image with another image. Follow the image keyword with the composite operator, image location, image size, and filename:

['-draw'; 'image Over 100,100 225,225 image.jpg']

You can use 0,0 for the image size, which means to use the actual dimensions found in the image header. Otherwise, it will be scaled to the given dimensions.

Use **text** to annotate an image with text. Follow the text coordinates with a string. If the string has embedded spaces, enclose it in double quotes. Optionally you can include the image filename, type, width, height, or other image attribute by embedding special format character. See **-comment** for details.

For example,

['-draw'; 'text 100,100 ""%wx%h""']

annotates the image with 512x480 for an image whose width is 512 and height is 480.

If the first character of *string* is @, the text is read from a file titled by the remaining characters in the string.

Rotate rotates subsequent shape primitives and text primitives about the origin of the main image. If the **-region** option precedes the **-draw** option, the origen for transformations is the upper left corner of the region.

Translate translates them.

Scale scales them.

SkewX and **SkewY** skew them with respect to the origen of the main image or the region.

The transformations modify the current affine matrix, which is initialized from the initial affine matrix defined by the **-affine** option. Transformations are cumulative within the **-draw** option. The initial affine matrix is not affected; that matrix is only changed by the appearance of another **-affine** option. If another **-draw** option appears, the current affine matrix is reinitialized from the initial affine matrix.

Use **color** to change the color of a pixel to the fill color (see **-fill**). Follow the pixel coordinate with a method:

point
replace
floodfill
filltoborder
reset

Consider the target pixel as that specified by your coordinate. The **point** method recolors the target pixel. The **replace** method recolors any pixel that matches the color of the target pixel. **Floodfill** recolors any pixel that matches the color of the target pixel and is a neighbor, whereas **filltoborder** recolors any neighbor pixel that is not the border color. Finally, **reset** recolors all pixels.

You can set the primitive color, font, and font bounding box color with **-fill**, **-font**, and **-box** respectively. Options are processed in the **argv** order so be sure to use these options *before* the **-draw** option.

-edge <*radius*>
detect edges within an image

-emboss
emboss an image

-encoding <*type*>
specify the text encoding

Choose from *AdobeCustom, AdobeExpert, AdobeStandard, AppleRoman, BIG5, GB2312, Latin 2, None, SJIScode, Symbol, Unicode, Wansung.*

-enhance
apply a digital filter to enhance a noisy image

-equalize
perform histogram equalization to the image

-fill <*color*>
color to use when filling a graphic primitive

The color is specified using the format described in the "Color Names" section of *X(1).*

See **-draw** for further details.

-filter <*type*>
use this type of filter when resizing an image

Use this option to affect the resizing operation of an image (see **-geometry**). Choose from these filters:

Point
Box
Triangle
Hermite
Hanning
Hamming
Blackman

Gaussian
Quadratic
Cubic
Catrom
Mitchell
Lanczos
Bessel
Sinc

The default filter is **Lanczos**

-flip create a "mirror image"

reflect the scanlines in the vertical direction.

-flop create a "mirror image"

reflect the scanlines in the horizontal direction.

-font *<name>*
use this font when annotating the image with text

You can tag a font to specify whether it is a PostScript, TrueType, or OPTION1 font. For example, Arial.ttf is a TrueType font, ps:helvetica is PostScript, and x:fixed is OPTION1.

-foreground *<color>*
define the foreground color

The color is specified using the format described in the "Color Names" section of *X(1).*

-frame *<width>x<height>+<outer bevel width>+<inner bevel width>*
surround the image with an ornamental border

See **-geometry** for details about the geometry specification. The **-frame** option is not affected by the **-gravity** option.

The color of the border is specified with the **-mattecolor** command line option.

-frame
include the X window frame in the imported image

-fuzz *<distance>{%}*
colors within this distance are considered equal

A number of algorithms search for a target color. By default the color must be exact. Use this option to match colors that are close to the target color in RGB space. For example, if you want to automatically trim the edges of an image with **-trim** but the image was scanned and the target background color may differ by a small amount. This option can account for these differences.

The *distance* can be in absolute intensity units or, by appending "*%*", as a percentage of the maximum possible intensity.

-gamma *<value>*
level of gamma correction

The same color image displayed on two different workstations may look different due to differences in the display monitor. Use gamma correction to adjust for this color

difference. Reasonable values extend from **0.8** to **2.3**.

You can apply separate gamma values to the red, green, and blue channels of the image with a gamma value list delineated with slashes (e.g., **1.7**/**2.3**/**1.2**).

-Gaussian *<radius>x<sigma>*
blur the image with a Gaussian operator

Use the given radius and standard deviation (sigma).

-geometry *<width>x<height>{+-}<x>{+-}<y>{%}{@} {!}{<}{>}*
preferred size and location of the Image window.

By default, the window size is the image size and the location is chosen by you when it is mapped.

By default, the width and height are maximum values. That is, the image is expanded or contracted to fit the width and height value while maintaining the aspect ratio of the image. *Append an exclamation point to the geometry to force the image size to exactly the size you specify.* For example, if you specify 640x480! the image width is set to 640 pixels and height to 480.

If only the width is specified, the width assumes the value and the height is chosen to maintain the aspect ratio of the image. Similarly, if only the height is specified (e.g., -geometry x256), the width is chosen to maintain the aspect ratio.

To specify a percentage width or height instead, append %. The image size is multiplied by the width and height percentages to obtain the final image dimensions. To increase the size of an image, use a value greater than 100 (e.g. 125%). To decrease an image's size, use a percentage less than 100.

Use @ to specify the maximum area in pixels of an image.

Use > to change the dimensions of the image *only* if its width or height exceeds the geometry specification. < resizes the image *only* if both of its dimensions are less than the geometry specification. For Example, if you specify '640x480>' and the image size is 256x256, the image size does not change. However, if the image is 512x512 or 1024x1024, it is resized to 480x480. Enclose the geometry specification in quotation marks to prevent the < or > from being interpreted by your shell as a file redirection.

-geometry is synonymous with **-resize** and specifies the size of the output image. The offsets, if present, are ignored.

-gravity *<type>*
direction primitive gravitates to when annotating the image.

Choices are: NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast.

The direction you choose specifies where to position the text or other graphic primitive when annotating the image. For example *Center* gravity forces the text to be centered within the image. By default, the image gravity is *NorthWest*. See **-draw** for more details about graphic primitives.

The **-gravity** option is also used in concert with the **-geometry** option and other options that take <**geometry**> as a parameter, such as the **-crop** option. See **-geometry** for details of how the **-gravity** option interacts with the <**x**> and <**y**> parameters of a geometry specification.

-implode <*factor*>
implode image pixels about the center. It is a real number.

-interlace <*type*>
the type of interlacing scheme

Choices are: **None, Line, Plane,** or **Partition**. The default is **None**.

This option is used to specify the type of interlacing scheme for raw image formats such as **RGB** or **YUV**.

None means do not interlace (RGBRGBRGBRGBRGBRGB...),

Line uses scanline interlacing (RRR...GGG...BBB...RRR...GGG...BBB...), and

Plane uses plane interlacing (RRRRRR...GGGGGG...BBBBBB...).

Partition is like plane except the different planes are saved to individual files (e.g. image.R, image.G, and image.B).

Use **Line** or **Plane** to create an **interlaced PNG** or **GIF** or **progressive JPEG** image.

-lat <*radius*>x<*sigma*>{+-}<*offset*>{%}
perform local adaptive thresholding

Perform local adaptive thresholding using the specified radius, sigma, and offset. The offset is a distance in sample space from the mean, as an absolute integer ranging from 0 to the maximum sample value or as a percentage. For reasonable results, radius should be larger than sigma. Use a radius of 0 to have the method select a suitable radius.

-level <*value*>
adjust the level of image contrast

Give three point values delineated with commas: black, mid, and white (e.g. 10,1.0,65000). The white and black points range from 0 to MaxRGB and mid ranges from 0 to 10.

-linewidth
the line width for subsequent draw operations

-mask <*filename*>
Specify a clipping mask

The image read from the file is used as a clipping mask. It must have the same dimensions as the image being masked.

If the mask image contains an opacity channel, the opacity of each pixel is used to define the mask. Otherwise, the intensity (gray level) of each pixel is used.

Use **+mask** to remove the clipping mask.

It is not necessary to use **-clip** to activate the mask; **-clip** is implied by **-mask**.

-median *<radius>*
apply a median filter to the image

-monochrome
transform the image to black and white

-negate
replace every pixel with its complementary color

The red, green, and blue intensities of an image are negated. White becomes black, yellow becomes blue, etc. Use **+negate** to only negate the grayscale pixels of the image.

-noise *<radius|type>*
add or reduce noise in an image

The principal function of noise peak elimination filter is to smooth the objects within an image without losing edge information and without creating undesired structures. The central idea of the algorithm is to replace a pixel with its next neighbor in value within a pixel window, if this pixel has been found to be noise. A pixel is defined as noise if and only if this pixel is a maximum or minimum within the pixel window.

Use **radius** to specify the width of the neighborhood.

Use **+noise** followed by a noise type to add noise to an image. Choose from these noise types:

Uniform
Gaussian
Multiplicative
Impulse
Laplacian
Poisson

-normalize
transform image to span the full range of color values

This is a contrast enhancement technique.

-opaque *<color>*
change this color to the pen color within the image

The color is specified using the format described in the "Color Names" section of *X(1)*.

-paint *<radius>*
simulate an oil painting

Each pixel is replaced by the most frequent color in a circular neighborhood whose width is specified with *radius*.

-raise *<width>x<height>*
lighten or darken image edges

This will create a 3-D effect. See **-geometry** for details details about the geometry specification. Offsets are not used.

Use **-raise** to create a raised effect, otherwise use **+raise**.

-region <*width*>x<*height*>{+-}<*x*>{+-}<*y*>
apply options to a portion of the image

The *x* and *y* offsets are treated in the same manner as in **-crop**.

-resize <*width*>x<*height*>{%}{@}{!}{<}{>}
resize an image

This is an alias for the **-geometry** option and it behaves in the same manner. If the **-filter** option precedes the **-resize** option, the specified filter is used.

There are some exceptions:

When used as a *composite* option, **-resize** conveys the preferred size of the output image, while **-geometry** conveys the size and placement of the *composite image* within the main image.

When used as a *montage* option, **-resize** conveys the preferred size of the montage, while **-geometry** conveys information about the tiles.

-roll {+-}<*x*>{+-}<*y*>
roll an image vertically or horizontally

See **-geometry** for details the geometry specification. The *x* and *y* offsets are not affected by the **-gravity** option.

A negative *x* offset rolls the image left-to-right. A negative *y* offset rolls the image top-to-bottom.

-rotate <*degrees*>{<}{>}
apply Paeth image rotation to the image

Use > to rotate the image only if its width exceeds the height. < rotates the image *only* if its width is less than the height. For example, if you specify -rotate "-90>" and the image size is 480x640, the image is not rotated. However, if the image is 640x480, it is rotated by -90 degrees. If you use > or <, enclose it in quotation marks to prevent it from being misinterpreted as a file redirection.

Empty triangles left over from rotating the image are filled with the color defined as **background** (class **backgroundColor**). See *X(1)* for details.

-sample <*geometry*>
scale image with pixel sampling

See **-geometry** for details about the geometry specification. **-sample** ignores the **-filter** selection if the **-filter** option is present. Offsets, if present in the geometry string, are ignored, and the **-gravity** option has no effect.

-sampling_factor <*horizontal_factor*>x<*vertical_factor*>
sampling factors used by JPEG or MPEG-2 encoder and YUV decoder/encoder.

This option specifies the sampling factors to be used by the JPEG encoder for chroma downsampling. If this option is omitted, the JPEG library will use its own default values. When reading or writing the YUV format and when writing the M2V (MPEG-2) format, use **-sampling_factor 2x1** to specify the 4:2:2 downsampling method.

-scale <*geometry*>
scale the image.

See **-geometry** for details about the geometry specification. **-scale** uses a simpler, faster algorithm, and it ignores the **-filter** selection if the **-filter** option is present. Offsets, if present in the geometry string, are ignored, and the **-gravity** option has no effect.

-seed <*value*>
pseudo-random number generator seed value

-segment <*cluster threshold*>x<*smoothing threshold*>
segment an image

Segment an image by analyzing the histograms of the color components and identifying units that are homogeneous with the fuzzy c-means technique.

Specify *cluster threshold* as the number of pixels in each cluster must exceed the the cluster threshold to be considered valid. *Smoothing threshold* eliminates noise in the second derivative of the histogram. As the value is increased, you can expect a smoother second derivative. The default is 1.5. See "Image Segmentation", below, for details.

-shade <*azimuth*>x<*elevation*>
shade the image using a distant light source

Specify *azimuth* and *elevation* as the position of the light source. Use **+shade** to return the shading results as a grayscale image.

-sharpen <*radius*>x<*sigma*>
sharpen the image

Use a Gaussian operator of the given radius and standard deviation (sigma).

-shave <*width*>x<*height*>
shave pixels from the image edges

Specify the width of the region to be removed from both sides of the image and the height of the regions to be removed from top and bottom.

-shear <*x degrees*>x<*y degrees*>
shear the image along the X or Y axis

Use the specified positive or negative shear angle.

Shearing slides one edge of an image along the X or Y axis, creating a parallelogram. An X direction shear slides an edge along the X axis, while a Y direction shear slides an edge along the Y axis. The amount of the shear is controlled by a shear angle. For X direction shears, *x degrees* is measured relative to the Y axis, and similarly, for Y direction shears *y degrees* is measured relative to the X axis.

Empty triangles left over from shearing the image are filled with the color defined as **background** (class **backgroundColor**). See *X(1)* for details.

-solarize <*factor*>
negate all pixels above the threshold level

Specify *factor* as the percent threshold of the intensity (0 - 99.9%).

This option produces a *solarization* effect seen when exposing a photographic film to light during the development process.

-spread <*amount*>
displace image pixels by a random amount

Amount defines the size of the neighborhood around each pixel to choose a candidate pixel to swap.

-stroke <*color*>
color to use when stroking a graphic primitive

The color is specified using the format described in the "Color Names" section of *X(1)*.

See **-draw** for further details.

-strokewidth <*value*>
set the stroke width

See **-draw** for further details.

-swirl <*degrees*>
swirl image pixels about the center

Degrees defines the tightness of the swirl.

-threshold <*value*>
threshold the image Create a bi-level image such that any pixel intensity that is equal or exceeds the threshold is reassigned the maximum intensity otherwise the minimum intensity.

-tile <*geometry*>
layout of images [*montage*]

-transform
transform the image

This option applies the transformation matrix from a previous **-affine** option.

```
mogrify(img, '-affine 2,2,-2,2,0,0 -transform bird.ppm');
```

-treedepth <*value*>
tree depth for the color reduction algorithm

Normally, this integer value is zero or one. A zero or one tells display to choose an optimal tree depth for the color reduction algorithm

An optimal depth generally allows the best representation of the source image with the fastest computational speed and the least amount of memory. However, the default depth is inappropriate for some images. To assure the best representation, try values between 2 and 8 for this parameter. Refer to quantize for more details.

The **-colors** or **-monochrome** option is required for this option to take effect.

-trim trim an image

This option removes any edges that are exactly the same color as the corner pixels. Use **-fuzz** to make **-trim** remove edges that are nearly the same color as the corner pixels.

-unsharp <*radius*>x<*sigma*>
sharpen the image with an unsharp mask operator

Use the given radius and standard deviation (sigma).

-wave <*amplitude*>x<*wavelength*> alter an image along a sine wave

Specify *amplitude* and *wavelength* of the wave.

IMAGE SEGMENTATION

Use **-segment** to segment an image by analyzing the histograms of the color components and identifying units that are homogeneous with the fuzzy c-means technique. The scale-space filter analyzes the histograms of the three color components of the image and identifies a set of classes. The extents of each class is used to coarsely segment the image with thresholding. The color associated with each class is determined by the mean color of all pixels within the extents of a particular class. Finally, any unclassified pixels are assigned to the closest class with the fuzzy c-means technique.

The fuzzy c-Means algorithm can be summarized as follows:

Build a histogram, one for each color component of the image.

For each histogram, successively apply the scale-space filter and build an interval tree of zero crossings in the second derivative at each scale. Analyze this scale-space "fingerprint" to determine which peaks or valleys in the histogram are most predominant.

The fingerprint defines intervals on the axis of the histogram. Each interval contains either a minima or a maxima in the original signal. If each color component lies within the maxima interval, that pixel is considered "classified" and is assigned an unique class number.

Any pixel that fails to be classified in the above thresholding pass is classified using the fuzzy c-Means technique. It is assigned to one of the classes discovered in the histogram analysis phase.

The fuzzy c-Means technique attempts to cluster a pixel by finding the local minima of the generalized within group sum of squared error objective function. A pixel is assigned to the closest class of which the fuzzy membership has a maximum value.

For additional information see: *Young Won Lim, Sang Uk Lee*, "**On The Color Image Segmentation Algorithm Based on the Thresholding and the Fuzzy c-Means Techniques**", Pattern Recognition, Volume 23, Number 9, pages 935-952, 1990.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

imread, imwrite, imfinfo

NAME

normal − normalize array to a given range

SYNOPSIS

N = normal(M, upper, lower)

PARAMETERS

M array

N normalized array

upper and **lower**
range limits of N. *lower defaults* to 0. *upper defaults* to 1.

DESCRIPTION

Function **normal** normalizes a matrix *M* to occupy *lower-upper* range. The intent of this function is to be a shortcut to a simple but often used operation.

N = normal(M) normalizes *N* to 0-1 range.

N = normal(M,upper) normalizes *N* to 0-upper range.

N = normal(M,lower,upper) normalizes *N* to lower-upper range.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

Img = gray_imread('gra.jpg');
mini(Img)
maxi(Img)
N = normal(Img,255);
mini(N)
maxi(N)

chdir(initial_dir);
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

imshow

NAME

ntsc2rgb − converts from YIQ to RGB colorspace

SYNOPSIS

RGB = ntsc2rgb(YIQ)

PARAMETERS

RGB A RGB image (M x N x 3) in 0-1 range or a RGB colormap (M x 3) in 0-1 range

YIQ A YIQ image (M x N x 3) in 0-1 range or a YIQ colormap (M x 3) in 0-1 range

DESCRIPTION

ntsc2rgb(YIQ) converts an YIQ image or colormap from YIQ to RGB colorspace. The YIQ model is used in NTSC and European TV's. It is useful for b&w and color compatibility, since the cromaticity (I and Q) and luminance (Y) are conveniently isolated. **ntsc2rgb(YIQ)** uses the following operation to convert each YIQ triplet:

```
! R !   !  1.000  0.956    0.621   !   ! Y !
! G ! = !  1.000  -0.272   -0.647  ! * ! I !
! B !   !  1.000  -1.105   1.702   !   ! Q !
```

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

rgb = imread('tru.jpg');   // RGB colorspace 0-1
imshow(rgb);
yiq = rgb2ntsc(rgb);       // YIQ colorspace 0-1
yiq(:,:,2) = yiq(:,:,2) /4; // Decrease chromaticity
yiq(:,:,3) = yiq(:,:,3) /4;
rgb = ntsc2rgb(yiq);
imshow(rgb);

chdir(initial_dir);
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

rgb2ntsc, im2gray

NAME

percol − test if binary image is percolated

SYNOPSIS

p = percol(img [,direction])

PARAMETERS

img an image in which 0 is considered background, while positive values are considered object (also called site).

direction

1 if percolation is to be tested horizontally; 0 if it is to be tested vertically

p 1 if image is percolated; 0 otherwise.

DESCRIPTION

Function **percol** tests if an image is 4-connected from side to side. This is useful for analysing images comming from site percolation simulations. The **percol** routine has the advantage to be implemented in the C language.

EXAMPLE

```
p=0.02;
a = 1*(rand(10,10) <= p)
while (~percol(a))   // test the image if one side connects to the other
  p = p+0.02;
  a = 1*(rand(10,10) <= p);
end

// now, surely, the image is connected side-to-side (i.e., it has percolated)
xbasc();
imshow(1-a,2); // 1 is displayed as black
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

xgetmouse, xclick, locate

NAME

perim − perimeter of binary shape

SYNOPSIS

p = perim(Img)

PARAMETERS

Img binary array, 1 for object and 0 for background (double precision)

p approximate perimeter of the shape inside Img

DESCRIPTION

This is a simple routine to measure the perimeter of a 2D shape. It assumes border pixels as a polygonal line: adjacent border pixels has measure size 1 or sqrt(2).

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

Img = imread('star.bmp');
xbasc()
imshow(Img,2);
p = perim(Img)

chdir(initial_dir);
```

BUGS AND SHORTCOMINGS

Images are stored in double precision matrices. Hopefully, the next release will make usage of integer types.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

follow, bwborder, edge, im2bw

NAME

pifilter − pi filter for phased images

SYNOPSIS

filtered_image = pifilter(image[,filter_name])

PARAMETERS

image A gray image (normally a phased image).

filter_name

A low-pass filter. It can be ´mean' or 'low-pass' (see mkfilter). Default is 'mean'.

DESCRIPTION

Filters a phased image before unwrapping it. The purpose is to keep the phase jumps sharp: a convolution by a low-pass kernel would be catastrophic, because it would smooth these jumps, causing the demodulation algorithm to fail.

Principle:
1) We calculate the sinus and cosinus of the phase
2) We apply a convolution filtering on each component
3) We get the phase again with an atan

This function uses the mkfilter and imconv functions. Reasonable kernels are: 'low-pass' and 'mean'. A high-pass filter would make no sense as it would burst noise !

You can use pifilter for ordinary grayscale images to smooth them without loosing the edges. This filter is similar to the median filter, but will leave edges even stronger. However, the noise will more often not be removed; just attenuated.

EXAMPLE

```
a=gray_imread(SIPDIR+'images/photonics/pyramide_noisy.jpg');
xset("window",0);xbasc();
imshow(a);
improfile(a);//to visualise profiles
b=pifilter(a);
xset("window",1);xbasc();
imshow(b);
improfile(b);//jumps are still well visible and other parts are smoothed
```

REFERENCES:

"Techniques automatiques de raccordement de phase" by David Venet
Ecole Federale de Lausanne (1995-96)

AUTHORS

Jocelyn DRUEL <jocelyn.druel1@libertysurf.fr>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

unwrapl,unwrapp,mkfilter,imconv

NAME

poledetection − poles (or residues) detection in a wrapped phased image

SYNOPSIS

poles = poledetection(image[,threshold])

PARAMETERS

image The gray-level image containing the wrapped phase.

threshold
The minimum difference betwwen 2 pixels to consider that a phase jump occurs. Default=0.5

poles A matrix: elements with the value zero are consistent points, others (positive or negative) are residus (or poles).

DESCRIPTION

Ideally, phase unwrapping should not depend on the path followed to unwrapp. So we consider a small loop of 4 adjacents pixels. We decide that there's a phase jump if the difference between 2 neighbours is superior to a threshold (=0.5 by default) We count the number of jumps when we follow the loop clockwise and anticlockwise. Because of noise, we observe that these numbers are differents in certain locations of the image. The phase is called "inconsistent" in these points. Unwrapping process is very uncertain there.
These poles (or residues) can be marked so that the unwrapping process ignores them, or can be used to create branches in residue-cut tree algorithms.

EXAMPLE

```
stacksize(4e7);

pw=gray_imread(SIPDIR+'images/photonics/pyramide_noisy.jpg'); //phase wrapped
xset("window",0);xbasc();imshow(pw);

poles=abs(poledetection(pw));//locate residues, no matter of the sign
xset("window",1);xbasc();imshow(poles,[]);//bright pixels
// show places where phase jumps are very uncertains

//Avoid unwrapping these points:
mf=imvariance(pw);//calculate a "merit function"
//mark bad pixels as visited (merit function=3000):
//you can comment this line if you want to compare results with and without
//poledetection
mf(find(poles>0))=3000;

//phase unwrapping:
puw=unwrapp(pw,mf);
xset("window",2);xbasc();imshow(puw,[]);
```

REFERENCES

Phase unwrapping algorithms for radar interferometry: residue-cut, least-squares, and synthesis algorithms by Zebker and Lu, Journal of Optical Society Am. A, vol 15, N.3, March 1998

Satellite radar interferometry: two-dimensional phase unwrapping by Goldstein, Zebker, Werner in Radio Science, vol 23, number 4, pages 713-720, july-august 1988

AUTHORS

Jocelyn DRUEL <jocelyn.druel1@libertysurf.fr>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

unwrapl, unwrapp, imvariance

NAME

rgb2ntsc − converts from RGB to YIQ colorspace

SYNOPSIS

YIQ = rgb2ntsc(RGB)

PARAMETERS

RGB A RGB image (M x N x 3) in 0-1 range or a RGB colormap (M x 3) in 0-1 range

YIQ A YIQ image (M x N x 3) in 0-1 range or a YIQ colormap (M x 3) in 0-1 range

DESCRIPTION

rgb2ntsc(RGB) converts an RGB image or colormap from RGB to YIQ colorspace. The YIQ model is used in NTSC and European TV's. It is useful for b&w and color compatibility, since the cromaticity (I and Q) and luminance (Y) are conveniently isolated. **rgb2ntsc(RGB)** uses the following operation to convert each RGB triplet:

```
! Y !   !  0.299 0.587  0.114   !   ! R !
! I ! = !  0.596 -0.274 -0.322  ! * ! G !
! Q !   !  0.212 -0.523 0.311   !   ! B !
```

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

rgb = imread('tru.jpg');   // RGB colorspace, 0-1 range
imshow(rgb);
yiq = rgb2ntsc(rgb);       // YIQ colorspace, 0-1 range
yiq(:,:,2) = yiq(:,:,2) /4; // Decrease chromaticity
yiq(:,:,3) = yiq(:,:,3) /4;
rgb = ntsc2rgb(yiq);
imshow(rgb);

chdir(initial_dir);
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

ntsc2rgb, im2gray

NAME

sip_fftshift − swap arrays (used with fft)

SYNOPSIS

Y = sip_fftshift(X)

PARAMETERS

X and Y

double vectors

DESCRIPTION

sip_fftshift(X) swaps the left and right halves of X.

sip_fftshift is a substitute for mtlb_fftshift, a Scilab function which is in FRACLAB contribution. We provide it here, since its necessary for many image processing tasks.

REMARKS

fftshift is now present in Scilab CVS. sip_fftshift is provided only for those versions of scilab that do not have fftshift.

There are optional arguments not documented, as well as other features of this function that we don't cover in this manpage.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>, with help from Scilab Group <Scilab@inria.fr>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

fft, FRACLAB toolbox

NAME

skel − skeletonization, thinning, Medial Axis Transform

SYNOPSIS

[skl,dt,lbl] = skel(img [,side, algorithm])

PARAMETERS

img Binary image containing one or more binary shapes. (foreground == 1, background == 0), One-pixel-wide regions are ignored (temporary limitation).

side ´interior´ if only internal skeleton is desired (DEFAULT);

´exterior´ if only external skeleton is desired;

´both´ if the background and foreground skeleton must be computed at the same time.

algorithm

´fast euclidean´ (DEFAULT) will perform a fast O(n) algorithm using the euclidean metric. For large and thick shapes, there may be a few small errors, which are dispensible for most practical applications.

´exact euclidean´ will perform an exact euclidean algorithm that is very much slower.

skl The multiscale skeleton image. This is a grayscale image, which may be thresholded to yield a skeleton with varying levels of detail. The greater the threshold, the cleaner is the skeleton. A threshold level of 5 will give a usual skeleton similar to the one obtained by popular thinning methods.

dt The euclidean distance transform of the image. It has the squared euclidean distances of any point of the image to the object.

lbl Label image. This is the discrete Voronoi Diagram of the boundary pixels of the considered object. Is is a grayscale image indicating the region of influence of each boundary pixel.

DESCRIPTION

Function **skel** performs skeletonization (thinning) of a binary object. The resulting medial axis is multi-scale, meaning that it can be progressively pruned to eliminate detail. This pruning is done by thresholding the output skeleton image.

The algorithm computes skeletons that are guaranteed to be connected over all scales of simplification. The skeletons are computed using the euclidean metric. This has the advantage to produce high-quality, isotropic and well-centered skeletons in the shape. However the exact algorithm is computationally intensive.

The radius of the maximal balls associated with the skeleton points are stored in the distance transform output image.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');
xset('auto clear', 'on');

im=gray_imread('escher.png');
```

```
imshow(im,2);
[skl,dt,vor] = skel(im);

// Fine detail
sklt = (skl >= 5);
imshow(im+sklt,[]);

// Less detail
sklt = (skl >= 20);
imshow(im+sklt,[]);

// The Distance Transform
imshow(sqrt(dt),[]);

// The Influence/Voronoi diagram of each boundary pixel
imshow(vor+1,rand(maxi(vor)+1,3));   // each region maps to a random color

// Let's see if computation is really fast
big = mogrify(im,['-sample','1000x']);
size(big)
skl = skel(big);
imshow(big + (skl >= 50),[]);


xset('auto clear', 'off');
chdir(initial_dir);
```

REFERENCES

For the fast euclidean algorithm: "Multiscale Skeletons by Image Foresting Transform and its Application to Neuromorphometry", A.X. Falcao, L. da F. Costa, B.S. da Cunha, Pattern Recognition, 2002.

For the exact euclidean algorithm:

"Multiresolution shape representation without border shifting", Costa, LF and Estrozi, Electronics Letters, no. 21, vol. 35, pp. 1829-1830, 1999.

"Shape Analysis and Classification", L. da F. Costa and R.M. Cesar Jr., CRC Press.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

thin, reconstruction (not done yet...)

NAME

thin − thinning by border deletion

SYNOPSIS

out = thin(img)

PARAMETERS

img Binary image containing one or more binary shapes. (foreground == 1, background == 0),

out Internal skeleton, thinned version of the shapes in the input image.

DESCRIPTION

Function **thin** performs thinning of binary objects. It uses the Zhang-Suen, a de facto standard and simple technique. The resulting image, the skeleton, is not always connected and is very sensible to noise. It is also slower than a good skeletonization algorithm (see **skel**). For thin shapes, it should work faster and provide better quality. You will need some pruning criterium to eliminate spurs.

EXAMPLE

```
im=gray_imread(SIPDIR+'images/r.gif');
imshow(im,2);

skl = thin(im);

xbasc();
imshow(im+skl,[]);

// Quality is definitely inferior to that of good skeletonization
// methods, as in the following test

im=gray_imread(SIPDIR+'images/escher.png');
skl = thin(im);          // Ordinary thinning
xbasc();
xset('wdim',400,500);
subplot(1,2,1);
imshow(im+skl,[]);
xset('wdim',800,400);

skl = skel(im);          // multiscale euclidean skeletonization
subplot(1,2,2);
imshow(im+(skl >= 10),[]);
xset('wdim',800,400);
```

REFERENCES

"Practical Computer Vision using C", J. R. Parker, Wiley.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

skel

NAME

unfollow − converts a parametric contour into a binary image

SYNOPSIS

Img = unfollow(x,y,dims)

PARAMETERS

x and y
: vectors, storing the parametrized contour. They are truncated before calculation.

dims
: vector [row, column] storing the dimensions of the output Image. This has to be consistent with the size of the resulting contour.

Img
: binary array, 1 for object and 0 for background (double precision)

DESCRIPTION

Function **unfollow** plots a contour from its parametric representation into a binary image. This operation is the reverse of the *follow* function.

x and y both store the parametrized contour. That is, (x(i),y(i)) is a point of the contour, where the coordinate system is assumed as starting from bottom-left corner (0,0) to uper-right corner of the image. To get (x,y) coordinates from (row,col) matrix coordinates, use the transformation below:

```
x = col - 1
y = dims(1) - row
```

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

Img = imread('star.bmp');
xset('window',0);
xbasc()
imshow(Img,2);
[x,y] = follow(Img);
xbasc()
xsm = gsm(x,15);
ysm = gsm(y,15);
Img2=unfollow(xsm,ysm,size(Img));
xbasc()
imshow(Img2,2);

chdir(initial_dir);
```

BUGS AND SHORTCOMINGS

Images are stored in double precision matrices. Hopefully, the next release will make usage of integer types.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

follow, bwborder, gsm, curvature

NAME

unwrapl − unwraps phased images linearly

SYNOPSIS

unwrapped_phase = unwrapl(image[,threshold,step,direction])

PARAMETERS

unwrapped_phase
is a matrix containing the unwrapped phase. Its values are not in 0-1 range, but depend of the number of phase jumps which were made: You have to normalize it to 0-1 if you want to visualize it.

image The gray-level image containing the wrapped phase.

threshold
If the difference between the value of two succesive pixels is higher than the threshold, we consider that there's a jump of phase in the image. Default=0.5

step Try to be a little noise immune by not allowing another jump in the phase if the distance from the previous is less than the value of step. Default=0.

direction
"h" to scan the image row by row, "v" to scan columnwise. Default="h"

DESCRIPTION

Phased images are obtained in various interferometry domains. The phase (coded in gray levels) is represented as the altitude of each pixel. Because the phase is modulo (2*pi or 1 when working with gray levels), the absolute altitude is not known.

This function unwraps a phased image (gray levels) in the simplest manner: linearly. It works this way:
1) computes a linear matrix: 1st line is read from left to right, 2nd line is read from right to left, etc...
2) compares the gradient to a threshold (0.5 is the default): add or substract 1 to ensure phase continuity.
3) rebuilds a image matrix from the linear matrix
4) step: don't authorize 2 phase jumps which are too close from one another
5) direction: h (horizontal: default) or v (vertical)
6) jumps is the map containing the number of phase jumps

Very simple and quite fast algorithm. But very noise sensitive. Images to treat should be of excellent quality.

For those not familiar with phase unwrapping, I tried to write a very detailed example.

EXAMPLE

```
stacksize(1e7); // images are memory-expensive

pw=gray_imread(SIPDIR+'images/photonics/pyramide_wrapped.jpg'); //phase wrapped

xset("window",0); //create a 1st window to display the original image
xbasc();xselect();imshow(pw);
xtitle("original wrapped phase")

//we try to show the object in 3D:
//because of phase jumps, it's not very good
xset("window",1);//the best of all: in a 3rd window, show the object in 3D
```

```
xbasc();xselect();
//we take 1 point on 4 to draw the object (faster and more beautifull than
//drawing all the points)
plot3d1(1:4:size(pw,'r'),1:4:size(pw,'c'),pw(1:4:$,1:4:$));
xtitle("original wrapped phase in 3D")

//now we unwrap the phase linearly:
//In this example we don't need any additionnal parameters
//because the image is of good quality.
//be a little patient for this operation
puw=unwrapl(pw);//phase unwrapped

//and we show the result
xset("window",2);//show the unwrapped phase in 2D: we have to put it in the 0-1 range
//to display it properly
xbasc();xselect();imshow(normal(puw));
xtitle("unwrapped phase");

xset("window",3);//we can now show the original object in 3D
xbasc();xselect();
// Again, we take 1 point on 4 to draw the object
plot3d1(1:4:size(puw,'r'),1:4:size(puw,'c'),puw(1:4:$,1:4:$));
xtitle("unwrapped phase in 3D");
```

REFERENCES

An easy introduction to these problems can be found in
"Methods for 2-D phase unwrapping in Matlab" by Jiri Novak.

A more complete one:
"Phase unwrapping algorithms for radar interferometry: residue-cut, least-squares, and synthesis algorithms"
by Zebker & Lu (Journal of Optical Society America, vol 15, n3, march 98)

AUTHORS

Jocelyn DRUEL <jocelyn.druel1@libertysurf.fr>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

unwrapp, imvariance, imphase

NAME

unwrapp − unwraps phased images by path following

SYNOPSIS

[unwrapped_phase,jumps] = unwrapp(image,merit_function, ...
[line_start,column_start])

PARAMETERS

image The gray-level image containing the wrapped phase.

merit_function

The quality of each pixel is estimated: the result is stored in a "merit function" (mf). Some merit functions can be: laplacien, variance... A good quality pixel has a low merit function.

line_start, column_start

The starting point to begin the path: if not entered, the function will search the best point (=the point with the lowest merit function) in a sqarre around the center of the image.

DESCRIPTION

For an explanation on phased images, have a look at the help of the function unwrapl.

This function uses an algorithm of "path following" to unwrapp the phase: The idea is to unwrapp non ambiguous pixels first. Here are some details:
1) the quality of each pixel is estimated by a "merit function". Most often, the "merit function" will be variance (function "imvariance"), but you could try others functions like laplacien. A good quality pixel has a low merit function.
2) We begin from a point of good quality. It is called "integrator point".
3) we rely it to its best quality neighbour, which becomes the new "integrator point".
4) If the difference between phases of the 2 points is higher (or lower) than a threshold (=127.5 for 8bit images), then we consider there's a phase jump.
5) we continue until all points are treated.

How to ignore some points ? You can decide that some points are so unsure that you prefer to simply ignore them. This can be done by affecting them a merit function >= 3000.

Note about the "jumps" matrix: it is a matrix containing only the number of jumps needed to re-establish phase continuity: unwrapped_phase=image+256*jumps;

EXAMPLE

```
stacksize(4e7);

pw=imread(SIPDIR+'images/photonics/pyramide_wrapped.jpg'); //phase wrapped

mf=imvariance(pw);//calculate a "merit function"
//you could try also: mf=imconv(a,mkfilter('laplace1'));

// this will take a few minutes
[puw,jumps]=unwrapp(pw,mf);

xset("window",1);xbasc();xselect();imshow(normal(puw,1,0));
xtitle("unwrapped phase");
xset("window",2);xbasc();plot3d1(1:8:size(puw,'r'),1:8:size(puw,'c'),puw(1:8:$,1:8:$))
```

REFERENCES

David VENET: "techniques automatiques de raccordement de phase" Memoire de l'universite de Lausanne:

AUTHORS

Jocelyn DRUEL <jocelyn.druel1@libertysurf.fr>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

unwrapl, imvariance, imphase

NAME

watershed − image segmentation with markers

SYNOPSIS

w = watershed(img [,markers, nhood])

PARAMETERS

img Grayscale image array. Preferably the gradient of the image to segment.

markers

image with the markers (seeds). Each mark must have a unique label from 1 to N, where N is the number of marks. If ´markers´ is ´-1´ or omitted, then the regional minima of ´im´ will be taken as the markers.

nhood

A scalar. The connectivity to consider in the algorithm. May be 4 or 8, for 4-neighborhood or 8-neighborhood, resp.

w image with the watershed regions (catchment basins), each with a unique number, from 1 to N. The elements labeled 1 belong to the first watershed region, the elements labeled 2 belong to the second basin, and so on. If markers != -1 or is omitted, the regions will have the same label as the corresponding supplied markers.

DESCRIPTION

Function **watershed** computes the Watershed transform for image segmentation. This operation is also known as morphological sup-reconstruction. It is a region-growing algorithm which partitions the image into regions around each marker. Read the references or search the Internet for more theoretical information.

In the example we show a very useful application of this operator: separation of overlapping objects.

EXAMPLE

```
// Suppose we have an image of many round or oval objects, such as a
// microscope image of blood cells.  After thresholding the image, we
// end up with a binary image like this:

xset('auto clear', 'on');
a = gray_imread(SIPDIR + 'images/disks.bmp');
imshow(a,2);

//
// We want to make the computer count the number of cells in the image,
// but there are some circles that are overlapping, thus forming a single
// connected component. Watershed is classically used for separating
// mingled objects like these.
//
// First, calculate the distance transform:

a = 1-a;
d = bwdist(a);
d = normal(sqrt(d),255);   // normalize it to 0-255 range
imshow(d+1, hotcolormap(256));

//
```

```
// The latter command shows the distance transform in shades from
// black to red to yellow to white. The brighter the color, the
// greater the distance of a point to the background.
// If you have the ENRICO toolbox, you can nicely plot the distance
// transform in 3D using sadesurf. ENRICO is not necessary for this
// example, but anyway you may download it at:
//       http://www.weizmann.ac.il/~fesegre/scistuff.html
//
// Note that the peaks of the distance transform are in the middle of
// each blob. The idea is to run watershed segmentation using these
// peaks as markers. For this, we invert the distance transform so
// that the peaks become the regional minima:

d = 255 - d;
imshow(d+1, hotcolormap(256));

// Now we "and" the distance transform with the original image, so
// that the background remains dark.

d = d .* a;
imshow(d+1,hotcolormap(256));

// Finally, run watershed segmentation. It automatically detects the
// regional minima for us:

w = watershed(d);
imshow(w, rand(256,3));

// 'w' is an image with a unique number for each watershed region.
// The imshow with a random colormap displays each region with a
// unique arbitrary color. Note how the regions were correctly separated by
// watershed, except for the hardest cases. It is extremely easy to
// count the number of regions:

n = maxi(w) - 1      // 26 regions minus the background

// The computer found 25 regions, but there are 20, an error of about 20%
// Let's improve this result. In the cases with many overlapping
// circles, the result would be perfect if it weren't for the small
// spurious regions. These are much smaller than the circles,
// so we can safely eliminate the regions with less than 100 pixels:

w2 = w;
for i=1:n
  f = find(w==i);        // coordinates of i-th region
  if size(f,'*') < 100
    w2(f) = 26;          // merge small regions with the background
  end
end

imshow(w2, rand(256,3));   // note how the small regions are gone

// Now we count again, using a different way:
```

```
n = size( unique(w2), '*') - 1     // subtract 1 for the background

// Now it's 100% correct! We have an authomatic method wich is surprisingly
// robust. This is specially useful to deal with bigger images in large
// ammount.
//
// Enjoy!
//
// TIP #1: Another way to improve the results is to do a median
// filtering of the distance transform. This will remove many spurious
// minima. Use mogrify(img, ['-median', '2']). Slight Gaussian smoothing also
// works well.
//
// TIP #2: for grayscale image segmentation, calculate the image
// gradient before watershed. Use edge(img,'sobel',-1) for this.
//
// TIP #3: use xgetpixel in a for loop (or something similar) to select
// the seed pixels to be used with watershed segmentation.

xset('auto clear', 'off');
```

REFERENCES

This is the algorithm we used:
"The Image Foresting Tranform: Theory, Algorithms, and Applications" A.X. Falcao, R.A. Lotufo, and J.Stolfi, IEEE T. Pattern Analysis and Machine Intelligence, (accepted for publication).
The IFT home page and the GIFT free software: http://www.ic.unicamp.br/~afalcao/ift.html
The original algorithm certainly that of Vincent & Soille, although it differs from the one we used in SIP/Animal:

"Watersheds in digital spaces: an efficient algorithm based on immersion simulations." IEEE T. Pattern Analysis and Machine Intelligence, 1991.

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

edge, bwdist, xgetpixel, mogrify (-segment option), im2bw, skel

NAME

xgetpixel − gets the pixel coordinates from an image

SYNOPSIS

p = xgetpixel(arg [,winno])

PARAMETERS

arg the number of rows in the considered image, or the image itself, depending on what's more convenient to the user.

winno the number of the scilab graphical window. Defaults to the current active window.

p the (row,col) coordinate of the pixel that have been clicked on the image.

DESCRIPTION

Function **xgetpixel** interactively shows the (row,col) position of the mouse cursor over an image ploted into scilab graphic.

EXAMPLE

```
initial_dir = PWD;
chdir (SIPDIR + 'images');

[im,map] = imread('example.bmp');
xbasc()
imshow(im,map);

p = xgetpixel(im)

chdir(initial_dir);
```

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

xgetmouse, xclick, locate

NAME

xor − exclusive or

SYNOPSIS

C = xor(A,B)

Input A, B: input arrays

Output

C: resulting boolean array

DESCRIPTION

"xor" is a simple convenience function to calculate the exclusive OR logical operator on the corresponding elements of the input arrays. The resulting element is %f if either the corresponding elements in A or in B are nonzero, but not both.

EXAMPLE

A = [0 0 %pi %eps] B = [0 -6.9 0 1]

C = xor(A,B) C = | F T T F |

AUTHORS

Ricardo Fabbri <rfabbri@if.sc.usp.br>

AVAILABILITY

The latest version of the Scilab Image Processing toolbox can be found at

http://siptoolbox.sourceforge.net

SEE ALSO

and, or