

# Package ‘EBImage’

March 26, 2013

**Version** 4.0.0

**Date** 2012-09-27

**Title** Image processing toolbox for R

**Author** Gregoire Pau, Andrzej Oles, Mike Smith, Oleg Sklyar, Wolfgang Huber

**Maintainer** Gregoire Pau <paug@gene.com>

**Depends**

**Imports** methods, graphics, stats, abind, utils, tiff, jpeg, png

**Description** EBImage is an R package which provides general purpose functionality for the reading, writing, processing and analysis of images. Furthermore, in the context of microscopy based cellular assays, EBImage offers tools to transform the images, segment cells and extract quantitative cellular descriptors.

**License** LGPL

**LazyLoad** true

**biocViews** Visualization

## R topics documented:

bwlabel . . . . .	2
channel . . . . .	3
Combine . . . . .	4
computeFeatures . . . . .	5
display . . . . .	8
distmap . . . . .	9
drawCircle . . . . .	10
EBImage . . . . .	11
EBImage-deprecated . . . . .	13
fillHull . . . . .	14
filter2 . . . . .	15
floodFill . . . . .	16
gblur . . . . .	17
Image . . . . .	18
io . . . . .	20
medianFilter . . . . .	22

morphology . . . . .	23
normalize . . . . .	25
ocontour . . . . .	25
paintObjects . . . . .	26
propagate . . . . .	27
resize . . . . .	29
rmObjects . . . . .	31
stackObjects . . . . .	32
thresh . . . . .	33
tile . . . . .	34
transpose . . . . .	35
watershed . . . . .	36
<b>Index</b>	<b>38</b>

---

bwlabel	<i>Binary segmentation</i>
---------	----------------------------

---

### Description

Labels connected (connected sets) objects in a binary image.

### Usage

bwlabel(x)

### Arguments

x                      An Image object or an array. x is considered as a binary image, whose pixels of value 0 are considered as background ones and other pixels as foreground ones.

### Details

All pixels for each connected set of foreground (non-zero) pixels in x are set to an unique increasing integer, starting from 1. Hence,  $\max(x)$  gives the number of connected objects in x.

### Value

An Grayscale Image object or an array, containing the labelled version of x.

### Author(s)

Gregoire Pau, 2009

### See Also

[link{computeFeatures}](#), [link{propagate}](#), [link{watershed}](#), [link{paintObjects}](#) }

**Examples**

```

## simple example
x = readImage(system.file('images', 'shapes.png', package='EBImage'))
x = x[110:512,1:130]
if (interactive()) display(x, title='Binary')
y = bwlabel(x)
if (interactive()) display(normalize(y), title='Segmented')

## read nuclei images
x = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
if (interactive()) display(x)

## computes binary mask
y = thresh(x, 10, 10, 0.05)
y = opening(y, makeBrush(5, shape='disc'))
if (interactive()) display(y, title='Cell nuclei binary mask')

## bwlabel
z = bwlabel(y)
if (interactive()) display(normalize(z), title='Cell nuclei')
nbnuclei = apply(z, 3, max)
cat('Number of nuclei=', paste(nbnuclei, collapse=', '), '\n')

## recolor nuclei in colors
cols = c('black', sample(rainbow(max(z))))
zrainbow = Image(cols[1+z], dim=dim(z))
if (interactive()) display(zrainbow, title='Cell nuclei (recolor)')

```

channel

*Color and image color mode conversions***Description**

channel handles color space conversions between image modes. rgbImage combines Grayscale images into a Color one.

**Usage**

```

channel(x, mode)
rgbImage(red, green, blue)

```

**Arguments**

x	An Image object or an array.
mode	A character value specifying the target mode for conversion. See Details.
red, green, blue	Image objects in Grayscale color mode or arrays of the same dimension. If missing, a black image will be used.

**Details**

Conversion modes:

- rgb Converts a Grayscale image or an array into a Color image, replicating RGB channels.
- gray, grey Converts a Color image into a Grayscale image, using uniform 1/3 RGB weights.
- red, green, blue Extracts the red, green or blue channel from a Color image. Returns a Grayscale image.
- asred, asgreen, asblue Converts a Grayscale image or an array into a Color image of the specified hue.

channel changes the pixel intensities, unlike colorMode which just changes the way that EBIImage should render an image,

**Value**

An Image object or an array.

**Author(s)**

Oleg Sklyar, <osklyar@ebi.ac.uk>

**See Also**

[colorMode](#)

**Examples**

```
x = readImage(system.file("images", "shapes.png", package="EBImage"))
if (interactive()) display(x)
y = channel(x, 'asgreen')
if (interactive()) display(y)

## rgbImage
x = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
y = readImage(system.file('images', 'cells.tif', package='EBImage'))
if (interactive()) display(x, title='Cell nuclei')
if (interactive()) display(y, title='Cell bodies')

cells = rgbImage(green=1.5*y, blue=x)
if (interactive()) display(cells, title='Cells')
```

---

Combine

*Combining images*

---

**Description**

Merges images to create image sequences.

**Usage**

```
combine(x, ..., along)
```

## Arguments

x	An Image object, an array, or a list of Image objects and arrays.
...	Image objects or arrays.
along	an optional numeric. See details.

## Details

The function combine uses abind to merge multi-dimensionnal arrays along the dimension specified by the value along.

If along is missing, a default value depending on the color mode of x is used. If x is a Grayscale image or an array, along is set to 3 and image objects are combined on this dimension. If x is a Color image, along is set to 4 and image objects are combined on this dimension, leaving room on the third dimension for color channels.

## Value

An Image object or an array.

## Author(s)

Gregoire Pau

## See Also

[Image](#)

## Examples

```
if (interactive()) {  
  ## combination of color images  
  lena = readImage(system.file("images", "lena-color.png", package="EBImage"))  
  x = combine(lena, flip(lena), flop(lena))  
  if (interactive()) display(x)  
  
  ## Blurred lenas  
  x = resize(lena, 128, 128)  
  xt = list()  
  for (t in seq(0.1, 5, len=9)) xt=c(xt, list(gblur(x, s=t)))  
  xt = combine(xt)  
  if (interactive()) display(xt, title='Blurred Lenas')  
}
```

---

computeFeatures

*Compute object features*

---

## Description

Computes morphological and texture features from image objects.

**Usage**

```

computeFeatures(x, ref, methods.noref=c("computeFeatures.moment", "computeFeatures.shape"),
  methods.ref=c("computeFeatures.basic", "computeFeatures.moment", "computeFeatures.haralick"),
  xname="x", refnames, properties=FALSE, expandRef=standardExpandRef, ...)
computeFeatures.basic(x, ref, properties=FALSE, basic.quantiles=c(0.01, 0.05, 0.5, 0.95, 0.99), xs, ...)
computeFeatures.shape(x, properties=FALSE, xs, ...)
computeFeatures.moment(x, ref, properties=FALSE, xs, ...)
computeFeatures.haralick(x, ref, properties=FALSE, haralick.nbins=32, haralick.scales=c(1, 2), xs, ...)
standardExpandRef(ref, refnames)

```

**Arguments**

x	An Image object or an array containing labelled objects. Labelled objects are pixel sets with the same unique integer value.
ref	A matrix or a list of matrices, containing the intensity values of the reference objects.
methods.noref	A character vector containing the function names to be called to compute features without reference intensities. Default is computeFeatures.moment and computeFeatures.shape.
methods.ref	A character vector containing the function names to be called to compute features with reference intensities. Default is computeFeatures.basic, computeFeatures.moment and computeFeatures.haralick.
xname	A character string naming the object layer. Default is x.
refnames	A character vector naming the reference intensity layers. Default are the names of ref, if present. If not, reference intensity layers are named using lower-case letters.
properties	A logical. If FALSE, the default, the function returns the feature matrix. If TRUE, the function returns feature properties.
expandRef	A function used to expand the reference images. Default is standardExpandRef. See Details.
basic.quantiles	A numerical vector indicating the quantiles to compute.
haralick.nbins	An integer indicating the number of bins using to compute the Haralick matrix. See Details.
haralick.scales	A integer vector indicating the number of scales to use to compute the Haralick features.
xs	An optional temporary object created by computeFeatures used for performance considerations.
...	Optional arguments passed to the feature computation functions.

**Details**

Features are named x.y.f, where x is the object layer, y the reference image layer and f the feature name. Examples include cell.dna.mean, indicating mean DNA intensity computed in the cell or nucleus.tubulin.cx, indicating the x center of mass of tubulin computed in the nucleus region.

The function computeFeatures computes sets of features. Features are organized in 4 sets, each computed by a different function. The function computeFeatures.basic computes spatial-independent statistics on pixel intensities:

- b.mean: mean intensity

- b.sd: standard deviation intensity
- b.mad: mad intensity
- b.q\*: quantile intensity

The function `computeFeatures.shape` computes features that quantify object shape:

- s.area: area size (in pixels)
- s.perimeter: perimeter (in pixels)
- s.radius.mean: mean radius (in pixels)
- s.radius.max: max radius (in pixels)
- s.radius.min: min radius (in pixels)

The function `computeFeatures.moment` computes features related to object image moments, which can be computed with or without reference intensities:

- m.cx: center of mass x (in pixels)
- m.cy: center of mass y (in pixels)
- m.majoraxis: elliptical fit major axis (in pixels)
- m.eccentricity: elliptical eccentricity defined by  $\sqrt{1 - \text{minoraxis}^2 / \text{majoraxis}^2}$ . Circle eccentricity is 0 and straight line eccentricity is 1.
- m.theta: object angle (in radians)

The function `computeFeatures.haralick` computes features that quantify pixel texture. Features are named according to Haralick's original paper.

### Value

If properties if FALSE (by default), `computeFeatures` returns a matrix of n cells times p features, where p depends of the options given to the function. Returns NULL if no object is present.

If properties if TRUE, `computeFeatures` returns a matrix of p features times 2 properties (translation and rotation invariance). Feature properties are useful to filter out features that may not be needed for specific tasks, e.g. cell position when doing cell classification.

### Author(s)

Gregoire Pau, <gregoire.pau@embl.de>, 2011

### References

R. M. Haralick, K Shanmugam and Its'Hak Deinstein (1979). *Textural Features for Image Classification*. IEEE Transactions on Systems, Man and Cybernetics.

### See Also

[bwlabel](#), [propagate](#)

**Examples**

```

## load and segment nucleus
y = readImage(system.file("images", "nuclei.tif", package="EBImage"))[,1]
x = thresh(y, 10, 10, 0.05)
x = opening(x, makeBrush(5, shape='disc'))
x = bwlabel(x)
if (interactive()) display(y, title="Cell nuclei")
if (interactive()) display(x, title="Segmented nuclei")

## compute shape features
fts = computeFeatures.shape(x)
fts

## compute features
ft = computeFeatures(x, y, xname="nucleus")
cat("median features are:\n")
apply(ft, 2, median)

## compute feature properties
ftp = computeFeatures(x, y, properties=TRUE, xname="nucleus")
ftp

```

display

*Image Display***Description**

Displays images using an interactive JavaScript viewer or R's built-in graphics capabilities.

**Usage**

```
display(x, title = paste(deparse(substitute(x))), method = c("browser", "raster"), frame, all = FALSE)
```

**Arguments**

x	an Image object or an array.
title	a character string used as a window title.
method	the method used to display images: either browser (default) for viewing images in a WWW browser, or raster for viewing them as raster graphics using R's native functions.
frame	a numeric indicating the frame number; only works in conjunction with method = "raster" and all = FALSE.
all	should all frames of a stacked image be displayed, or just a single frame?

**Details**

By default (method = "browser") images are displayed in a WWW browser using a JavaScript based image browser. Multiple windows or tabs can be opened in this way. Pressing H displays a list of available features along with corresponding mouse and keyboard actions.

If method = "raster" images are displayed using R's built-in rasterImage function. By default only the first frame of a stacked image is rendered; a different frame can be specified using frame. When all = TRUE all frames are rendered side by side and automatically positioned in a grid.



**Value**

Invisible NULL.

**Note**

For viewing images in a WWW browser a compatible browser with JavaScript enabled is required (e.g. Mozilla Firefox).

**Author(s)**

Andrzej Oles, <andrzej.oles@gmail.com>, 2012

**References**

Mozilla Firefox: <http://www.firefox.com>

**Examples**

```
## Display a single image
x = readImage(system.file("images", "lena-color.png", package="EBImage"))
if (interactive()) {
  display(x, "Lena")
  display(x, method = "raster")
}

## Display a thresholded sequence ...
y = readImage(system.file("images", "lena.png", package="EBImage"))
yt = list()
for (t in seq(0.1, 5, len=9)) yt=c(yt, list(gblur(y, s=t)))
yt = combine(yt)

## ... using the browser viewer ...
if (interactive()) display(yt, "Blurred Lenas")

## ... or using R's build-in raster functions
if (interactive()) display(resize(yt, 256, 256), method = "raster", all = TRUE)

## Display the last frame
if (interactive()) display(yt, method = "raster", frame = getNumberOfFrames(yt, type = "render"))
```

---

distmap

*Distance map transform*

---

**Description**

Computes the distance map transform of a binary image. The distance map is a matrix which contains for each pixel the distance to its nearest background pixel.

**Usage**

```
distmap(x, metric=c('euclidean', 'manhattan'))
```

**Arguments**

x	An Image object or an array. x is considered as a binary image, whose pixels of value 0 are considered as background ones and other pixels as foreground ones.
metric	A character indicating which metric to use, L1 distance (manhattan) or L2 distance (euclidean). Default is euclidean.

**Details**

A fast algorithm of complexity  $O(M*N*\log(\max(M,N)))$ , where (M,N) are the dimensions of x, is used to compute the distance map.

**Value**

An Image object or an array, with pixels containing the distances to the nearest background points.

**Author(s)**

Gregoire Pau, <gpau@ebi.ac.uk>, 2008

**References**

M. N. Kolountzakis, K. N. Kutulakos. Fast Computation of the Euclidean Distance Map for Binary Images, Infor. Proc. Letters 43 (1992).

**Examples**

```
x = readImage(system.file("images", "shapes.png", package="EBImage"))
if (interactive()) display(x)
dx = distmap(x)
if (interactive()) display(dx/10, title='Distance map of x')
```

---

drawCircle

*Draw a circle on an image.*

---

**Description**

Draw a circle on an image.

**Usage**

```
drawCircle(img, x, y, radius, col, fill=FALSE, z=1)
```

**Arguments**

img	An Image object or an array.
x, y, radius	numerics indicating the center and the radius of the circle.
col	A numeric or a character string specifying the color of the circle.
fill	A logical indicating whether the circle should be filled. Default is FALSE.
z	A numeric indicating on which frame of the image the circle should be drawn. Default is 1.

**Value**

An Image object or an array, containing the transformed version of img.

**Author(s)**

Gregoire Pau, 2010

**Examples**

```
## Simple white circle
x = matrix(0, nrow=300, ncol=300)
y = drawCircle(x, 100, 200, 47, col=1)
if (interactive()) display(y)

## Simple filled yellow circle
x = channel(y, 'rgb')
y = drawCircle(x, 200, 140, 57, col='yellow', fill=TRUE)
if (interactive()) display(y)
```

---

EBImage

*Package overview*

---

**Description**

EBImage is an image processing and analysis package for R. Its primary goal is to enable automated analysis of large sets of images such as those obtained in high throughput automated microscopy.

EBImage relies on the Image object to store and process images but also works on multi-dimensional arrays.

**Package content**

Image methods

- Image
- as.Image, is.Image, as.raster.Image
- colorMode, imageData
- getFrame, getNumberOfFrames

Image I/O, display

- readImage, writeImage
- display
- image

Spatial transforms

- resize, flip, flop, transpose
- rotate, translate, affine

Image segmentation, objects manipulation

- thresh, bwlabel
- watershed, propagate
- ocontour
- paintObjects, rmObjects, reenumerate

#### Image enhancement, filtering

- normalize
- filter2, gblur, medianFilter

#### Morphological operations

- makeBrush
- erode, dilate, opening, closing
- erodeGreyScale, dilateGreyScale, openingGreyScale, closingGreyScale
- whiteTopHatGreyScale, blackTopHatGreyScale, selfcomplementaryTopHatGreyScale
- distmap
- floodFill, fillHull

#### Colorspace manipulation

- rgbImage, channel

#### Image stacking, combining, tiling

- stackObjects
- combine
- tile, untile

#### Drawing on images

- drawCircle

#### Features extraction

- computeFeatures
- computeFeatures.basic, computeFeatures.moment, computeFeatures.shape, computeFeatures.haralick
- standardExpandRef

#### Deprecated

- blur, equalize
- drawtext, drawfont
- getFeatures, hullFeatures, zernikeMoments
- edgeProfile, edgeFeatures,
- haralickFeatures, haralickMatrix
- moments, cmoments, smoments, rmoments

**Authors**

Oleg Sklyar, <osklyar@ebi.ac.uk>, Copyright 2005-2007

Gregoire Pau, <gpau@ebi.ac.uk>

Wolfgang Huber, <huber@ebi.ac.uk>

Andrzej Oles, <andrzej.oles@gmail.com>

Mike Smith, <msmith@ebi.ac.uk>

European Bioinformatics Institute  
European Molecular Biology Laboratory  
Wellcome Trust Genome Campus  
Hinxton  
Cambridge CB10 1SD  
UK

The code of `propagate` is based on the CellProfiler with permission granted to distribute this particular part under LGPL, the corresponding copyright (Jones, Carpenter) applies.

The source code is released under LGPL (see the LICENSE file in the package root for the complete license wording).

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

See the GNU Lesser General Public License for more details. For LGPL license wording see <http://www.gnu.org/licenses/lgpl.html>

**Examples**

```
example(readImage)
example(display)
example(rotate)
example(propagate)
```

---

EBImage-deprecated      *EBImage deprecated functions*

---

**Description**

These following functions are deprecated and will be defunct in the next Bioconductor release.

**Usage**

```
blur(...)
equalize(...)
drawtext(...)
drawfont(...)
getFeatures(...)
```

hullFeatures(...)  
zernikeMoments(...)  
edgeProfile(...)  
edgeFeatures(...)  
haralickFeatures(...)  
haralickMatrix(...)  
moments(...)  
rmoments(...)  
smoments(...)  
cmoments(...)

### Arguments

...                      Deprecated arguments.

---

fillHull	<i>Fill holes in objects</i>
----------	------------------------------

---

### Description

Fill holes in objects.

### Usage

```
fillHull(x)
```

### Arguments

x                      An Image object or an array.

### Details

fillHull fills holes in the objects defined in x, where objects are sets of pixels with the same unique integer value.

### Value

An Image object or an array, containing the transformed version of x.

### Author(s)

Gregoire Pau, Oleg Sklyar; 2007

### See Also

[bwlabel](#)

**Examples**

```
x = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
if (interactive()) display(x)

y = thresh(x, 10, 10, 0.05)
if (interactive()) display(y, title='Cell nuclei')

y = fillHull(y)
if (interactive()) display(y, title='Cell nuclei without holes')
```

---

filter2

*2D Convolution Filter*

---

**Description**

Filters an image using the fast 2D FFT convolution product.

**Usage**

```
filter2(x, filter)
```

**Arguments**

x	An Image object or an array.
filter	An Image object or an array, with odd spatial dimensions. Must contain only one frame.

**Details**

Linear filtering is useful to perform low-pass filtering (to blur images, remove noise...) and high-pass filtering (to detect edges, sharpen images). The function `makeBrush` is useful to generate filters.

Data is reflected around borders.

If `x` contains multiple frames, the filter will be applied one each frame.

**Value**

An Image object or an array, containing the filtered version of `x`.

**Author(s)**

Gregoire Pau, <gpau@ebi.ac.uk>

**See Also**

[makeBrush](#), [convolve](#), [fft](#), [blur](#)

**Examples**

```
x = readImage(system.file("images", "lena-color.png", package="EBImage"))
if (interactive()) display(x, title='Lena')

## Low-pass disc-shaped filter
f = makeBrush(21, shape='disc', step=FALSE)
if (interactive()) display(f, title='Disc filter')
f = f/sum(f)
y = filter2(x, f)
if (interactive()) display(y, title='Filtered lena')

## High-pass Laplacian filter
la = matrix(1, nc=3, nr=3)
la[2,2] = -8
y = filter2(x, la)
if (interactive()) display(y, title='Filtered lena')
```

floodFill

*Region filling***Description**

Fill regions in images.

**Usage**

```
floodFill(x, pt, col, tolerance=0)
```

**Arguments**

x	An Image object or an array.
pt	Coordinates of the start filling point.
col	Fill color. This argument should be a numeric for Grayscale images and an R color for Color images.
tolerance	Color tolerance used during the fill.

**Details**

Flood fill is performed using the fast scan line algorithm. Filling starts at pt and grows in connected areas where the absolute difference of the pixels intensities (or colors) remains below tolerance.

**Value**

An Image object or an array, containing the transformed version of x.

**Author(s)**

Gregoire Pau, Oleg Sklyar; 2007



**Examples**

```
x = readImage(system.file("images", "shapes.png", package="EBImage"))
y = floodFill(x, c(67, 146), 0.5)
if (interactive()) display(y)
```

```
y = channel(y, 'rgb')
y = floodFill(y, c(48, 78), 'red')
y = floodFill(y, c(156, 52), 'orange')
if (interactive()) display(y)
```

```
x = readImage(system.file("images", "lena.png", package="EBImage"))
y = floodFill(x, c(226, 121), 1, tolerance=0.1)
if (interactive()) display(y)
```

---

gblur

*Low-pass Gaussian filter*

---

**Description**

Filters an image with a low-pass Gaussian filter.

**Usage**

```
gblur(x, sigma, radius = 2 * ceiling(3 * sigma) + 1)
```

**Arguments**

x	An Image object or an array.
sigma	A numeric denoting the standard deviation of the Gaussian filter used for blurring.
radius	The radius of the filter in pixels. Default is $2 * \text{ceiling}(3 * \text{sigma}) + 1$ .

**Details**

The Gaussian filter is created with the function `makeBrush`.

**Value**

An Image object or an array, containing the filtered version of x.

**Author(s)**

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2005-2007

**See Also**

[filter2](#), [makeBrush](#)

**Examples**

```
x = readImage(system.file("images", "lena.png", package="EBImage"))
if (interactive()) display(x)

y = gblur(x, sigma=8)
if (interactive()) display(y, title='gblur(x, sigma=8)')
```

Image

*Image class***Description**

The package EBImage uses the class Image to store and process images. Images are stored as multi-dimensional arrays containing the pixel intensities. The class Image extends the base class array and uses the colormode slot to store how the color information of the multi-dimensional data is handled.

The colormode slot could be either Grayscale or Color. Greyscale images containing an alpha channel are stored with colormode = Color. In both modes, the first two dimensions of the underlying array are understood to be the spatial dimensions of the image. In the Grayscale mode, the remaining dimensions contain other images. In the the Color mode, the third dimension contains color channels of the image: either the red, green, blue and, possibly, alpha channel, or the grey and alpha channels; the remaining dimensions contain other images.

All methods of the package EBImage work either with Image objects or multi-dimensional arrays but in the latter case, the color mode is assumed to be Grayscale.

**Usage**

```
Image(data, dim, colormode)
as.Image(x)
is.Image(x)
as.raster.Image(y, i)

colorMode(y)
colorMode(y) <- value

imageData(y)
imageData(y) <- value

getFrame(y, i, type='total')
getNumberOfFrames(y, type='total')
```

**Arguments**

data	A vector or array containing the pixel intensities of an image. If missing, a default 1x1 null array is used.
dim	A vector containing the final dimensions of an Image object. If missing, equals to dim(data).
colormode	A numeric or a character string containing the color mode which could be either Grayscale or Color. If missing, equals to Grayscale.
x	An R object.

y	An Image object or an array.
i	A numeric.
value	For colorMode, a numeric or a character string containing the color mode which could be either Grayscale or Color. For imageData, an Image object or an array.
type	A character string containing total or render. Default is total.

### Details

Depending of type, getNumberOfFrames returns the total number of frames contained in the object y or the number of renderable frames. The total number of frames is independent of the color mode and is equal to the product of all the dimensions except the two first ones. The number of renderable frames is equal to the total number of frames in the Grayscale color mode and is equal to the product of all the dimensions except the three first ones in the Color color mode.

getFrame returns the i-th frame contained in the image y. If type is total, the function is unaware of the color mode and returns an xy-plane. If render, the function returns the i-th image as shown by the display function.

### Value

Image and as.Image return a new Image object.

is.Image returns TRUE if x is an Image object and FALSE otherwise.

as.raster.Image coerces an Image object to a raster object. For stacked images the ith frame is returned (by default the first one).

colorMode returns the color mode of y and colorMode<- changes the color mode of y.

imageData returns the array contained in an Image object.

### Author(s)

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2005-2007

### See Also

[readImage](#), [writeImage](#), [display](#)

### Examples

```
s1 = exp(12i*pi*seq(-1, 1, length=300)^2)
y = Image(outer(Im(s1), Re(s1)))
if (interactive()) display(normalize(y))

x = Image(rnorm(300*300*3),dim=c(300,300,3), colormode='Color')
if (interactive()) display(x)

w = matrix(seq(0, 1, len=300), nc=300, nr=300)
m = abind::abind(w, t(w), along=3)
z = Image(m, colormode='Color')
if (interactive()) display(normalize(z))

y = Image(c('red', 'violet', '#ff51a5', 'yellow'), dim=c(71, 71))
if (interactive()) display(y)
```

```

## colorMode example
x = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
x = x[,1:3]
if (interactive()) display(x, title='Cell nuclei')
colorMode(x)=Color
if (interactive()) display(x, title='Cell nuclei in RGB')

```

io

*Image I/O***Description**

Read and write images from/to files and URLs.

**Usage**

```

readImage(files, type, all = TRUE, ...)
writeImage(x, files, type, quality = 100, bits.per.sample, compression = "none", ...)

```

**Arguments**

files	a character vector of file names or URLs.
type	image type (optional). Supported values are: jpeg, png, and tiff. If missing, file format is automatically determined by filename extension.
all	logical: when the file contains more than one image should all frames be read, or only the first one?
x	an Image object or an array.
bits.per.sample	a numeric scalar specifying the number of bits per sample (only for tiff files). Supported values are 8 and 16.
compression	the desired compression algorithm (only for tiff files). For a list of supported values consult the documentation of the <a href="#">writeTIFF</a> function from the <b>tiff</b> package.
quality	a numeric ranging from 1 to 100 (default) controlling the quality of the JPEG output.
...	arguments passed to the corresponding functions from the <b>jpeg</b> , <b>png</b> , and <b>tiff</b> packages.

**Details**

`readImage` loads all images from the `files` vector and returns them stacked into a single Image object containing an array of doubles ranging from 0 (black) to 1 (white). All images need to be of the same type and have the same dimensions and color mode. If `type` is missing, the appropriate file format is determined from filename extension. Color mode is determined automatically based on the number of channels. When the function fails to read an image it skips to the next element of the `files` vector issuing a warning message. Non-local files can be read directly from a valid URL.

`writeImage` writes images into files specified in `files`, were the number of files need to be equal 1 or the number of frames. Given an image containing multiple frames and a single filename either the whole stack is written into a single TIFF file, or each frame is saved to an individual JPEG/PNG file (for `files = "image.*"` frames are saved into `image-X.*` files, where X is the frame number less one; for an image containing `n` frames this results in filenames numbered from 0 to `n-1`).

When writing JPEG files the compression quality can be specified using `quality`. Valid values range from 100 (highest quality) to 1 (lowest quality). For TIFF files additional information about the desired number of bits per sample (`bits.per.sample`) and the compression algorithm (`compression`) can be provided. For a complete list of supported values please consult the documentation of the `tiff` package.

### Value

`readImage` returns a new Image object.

`writeImage` returns an invisible vector of filenames.

### Note

Image formats have a limited dynamic range (e.g. JPEG: 8 bit, TIFF: 16 bit) and `writeImage` may cause some loss of accuracy. In specific, writing 16 bit image data to formats other than TIFF will strip the 8 LSB. When writing TIFF files a dynamic range check is performed and an appropriate value of `bits.per.sample` is set automatically.

### Author(s)

Andrzej Oles, <[andrzej.oles@gmail.com](mailto:andrzej.oles@gmail.com)>, 2012

### See Also

[Image](#), [display](#), [readJPEG/writeJPEG](#), [readPNG/writePNG](#), [readTIFF/writeTIFF](#)

### Examples

```
## Read and display an image
f = system.file("images", "lena-color.png", package="EBImage")
x = readImage(f)
if (interactive()) display(x)

## Read and display a multi-frame TIFF
y = readImage(system.file("images", "nuclei.tif", package="EBImage"))
if (interactive()) display(y)

## Read an image directly from a remote location by specifying its URL
try({
  im = readImage("http://www-huber.embl.de/EBImage/ExampleImages/berlin.tif")
  if (interactive()) display(im, title = "Berlin Impressions")
})

## Convert a PNG file into JPEG
tempfile = tempfile("", , ".jpeg")
writeImage(x, tempfile, quality = 85)
cat("Converted '", f, "' into '", tempfile, "'.\n", sep="")

## Save a frame sequence
files = writeImage(y, tempfile("", , ".jpeg"), quality = 85)
cat("Files created: ", files, sep="\n")
```

---

`medianFilter`*2D constant time median filtering*

---

**Description**

Filters a 16-bit image using Perreault's modern constant time median filtering algorithm [1].

**Usage**

```
medianFilter(x, size, cacheSize=512)
```

**Arguments**

<code>x</code>	An Image object or an array.
<code>size</code>	The sidelength of the square median filter in units of pixels.
<code>cacheSize</code>	The L2 cache size of the system CPU in kB.

**Details**

Median filtering is useful as a smoothing technique, e.g. in the removal of speckling noise.

If `x` contains multiple frames, the filter will be applied on each frame.

**Value**

An Image object or an array, containing the filtered version of `x`.

**Author(s)**

Joseph Barry, <joseph.barry@embl.de>, 2012

**References**

[1] S. Perreault and P. Hebert, "Median Filtering in Constant Time", IEEE Trans Image Process 16(9), 2389-2394, 2007

**See Also**

[makeBrush](#), [fft](#), [gblur](#)

**Examples**

```
x = readImage(system.file("images", "nuclei.tif", package="EBImage"))
if (interactive()) display(x, title='Nuclei')
y = medianFilter(x, 5)
if (interactive()) display(y, title='Filtered nuclei')
```

---

 morphology
 

---



---

*Perform morphological operations on images*


---

**Description**

Functions to perform morphological operations on binary images.

**Usage**

```
dilate(x, kern)
erode(x, kern)
opening(x, kern)
closing(x, kern)
dilateGreyScale(x, kern)
erodeGreyScale(x, kern)
openingGreyScale(x, kern)
closingGreyScale(x, kern)
whiteTopHatGreyScale(x, kern)
blackTopHatGreyScale(x, kern)
selfcomplementaryTopHatGreyScale(x, kern)
```

```
makeBrush(size, shape=c('box', 'disc', 'diamond', 'gaussian', 'line'), step=TRUE, sigma=0.3, angle=45)
```

**Arguments**

x	An Image object or an array. x is considered as a binary image, whose pixels of value 0 are considered as background ones and other pixels as foreground ones.
kern	An Image object or an array, containing the structuring element. kern is considered as a binary image, whose pixels of value 0 are considered as background ones and other pixels as foreground ones.
size	A numeric containing the size of the brush, in pixels. If shape is line, this represents the length of the line.
shape	A character vector indicating the shape of the brush. Can be box, disc, diamond, gaussian or line. Default is box.
step	a logical indicating if the brush is binary. Default is TRUE. The argument is relevant only for the disc and diamond shapes.
sigma	An optional numeric containing the standard deviation of the Gaussian shape. Default is 0.3.
angle	An optional numeric containing the angle at which the line should be drawn. The angle is one between the top of the image and the line.

**Details**

dilate applies the mask positioning its centre over every background pixel (0), every pixel which is not covered by the mask is reset to foreground (1).

erode applies the mask positioning its centre over every foreground pixel (!=0), every pixel which is not covered by the mask is reset to background (0).

opening is an erosion followed by a dilation and closing is a dilation followed by an erosion. The same goes for the greyscale versions.

`dilateGreyScale` applies the mask positioning its centre over every pixel of the Image, the output value of the pixel is the maximum value of the Image covered by the mask.

`erodeGreyScale` applies the mask positioning its centre over every pixel of the Image, the output value of the pixel is the minimum value of the Image covered by the mask.

`whiteTopHatGreyScale` subtracts the opening of the Image from the Image

`blackTopHatGreyScale` subtracts the Image from the closing of the Image

`selfcomplementaryTopHatGreyScale` is the sum of a white top-hat and a black top-hat, simplified the difference between closing and opening of the Image

`makeBrush` generates brushes of various sizes and shapes that can be used as structuring elements.

Operations on greyscale images use an implementation of the Urbach-Wilkinson algorithm[1] and can only handle flat (i.e. binary) brushes.

### Value

`dilate`, `erode`, `opening`, `closing`, `dilateGreyScale`, `erodeGreyScale`, `openingGreyScale`, `closingGreyScale`, `whiteTopHatGreyScale`, `blackTopHatGreyScale` and `selfcomplementaryTopHatGreyScale` return the transformed Image object or array, after the corresponding morphological operation.

`makeBrush` generates a 2D matrix containing the desired brush.

### Author(s)

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2006 Ilia Kats, <ilia-kats@gmx.net>, 2012

### References

[1] E. R. Urbach and M.H.F. Wilkinson, "Efficient 2-D grayscale morphological transformations with arbitrary flat structuring elements", IEEE Trans Image Process 17(1), 1-8, 2008

### Examples

```
x = readImage(system.file("images", "shapes.png", package="EBImage"))
if (interactive()) display(x)
kern = makeBrush(5, shape='diamond')
if (interactive()) display(kern, title='Structuring element')
if (interactive()) display(erode(x, kern), title='Erosion of x')
if (interactive()) display(dilate(x, kern), title='Dilatation of x')

## makeBrush
x = makeBrush(100, shape='diamond')
if (interactive()) display(x, title="makeBrush(100, shape='diamond')")
x = makeBrush(100, shape='disc', step=FALSE)
if (interactive()) display(x, title="makeBrush(100, shape='disc', step=FALSE)")
x = makeBrush(100, shape='gaussian', sigma=10)
if (interactive()) display(2000*x, title="makeBrush(100, shape='gaussian', sigma=10)")
```



---

normalize	<i>Intensity values linear scaling</i>
-----------	--

---

**Description**

Linearly scale the intensity values of an image to a specified range.

**Usage**

```
normalize(x, separate=TRUE, ft=c(0,1))
```

**Arguments**

x	An Image object or an array.
separate	If TRUE, normalizes each frame separately.
ft	A numeric vector of 2 values, target minimum and maximum intensity values after normalization.

**Value**

An Image object or an array, containing the transformed version of x.

**Author(s)**

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2006-2007

**Examples**

```
x = readImage(system.file('images', 'shapes.png', package='EBImage'))
x = x[110:512,1:130]
y = bwlabel(x)
if (interactive()) display(x, title='Original')

print(range(y))
y = normalize(y)
print(range(y))

if (interactive()) display(y, title='Segmented')
```

---

ocontour	<i>Oriented contours</i>
----------	--------------------------

---

**Description**

Computes the oriented contour of objects.

**Usage**

```
ocontour(x)
```

**Arguments**

`x` An Image object or an array, containing objects. Only integer values are considered. Pixels of value 0 constitute the background. Each object is a set of pixels with the same unique integer value. Objects are assumed connected.

**Value**

A list of matrices, containing the coordinates of object oriented contours.

**Author(s)**

Gregoire Pau, <gpau@ebi.ac.uk>, 2008

**Examples**

```
x = readImage(system.file("images", "shapes.png", package="EBImage"))
x = x[1:120,50:120]
if(interactive()) display(x)
oc = ocontour(x)
plot(oc[[1]], type='l')
points(oc[[1]], col=2)
```

---

paintObjects

*Marks objects in images*

---

**Description**

This function marks objects in images.

**Usage**

```
paintObjects(x, tgt, opac=c(1, 1), col=c('red', NA))
```

**Arguments**

`x` An Image object in Grayscale color mode or an array containing object masks. Object masks are sets of pixels with the same unique integer value.

`tgt` An Image object or an array, containing the intensity values of the objects.

`opac` A numeric vector of two opacity values for drawing object boundaries and object bodies. Opacity ranges from 0 to 1, with 0 being fully transparent and 1 fully opaque.

`col` A character vector of two R colors for drawing object boundaries and object bodies. By default, object boundaries are painted in red while object bodies are not painted.

**Value**

An Image object or an array, containing the painted version of `tgt`.

**Author(s)**

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2006-2007

**See Also**

[bwlable](#), [watershed](#), [link{computeFeatures}](#)

**Examples**

```

## load images
nuc = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
cel = readImage(system.file('images', 'cells.tif', package='EBImage'))
img = rgbImage(green=cel, blue=nuc)
if (interactive()) display(img, title='Cells')

## segment nuclei
nmask = thresh(nuc, 10, 10, 0.05)
nmask = opening(nmask, makeBrush(5, shape='disc'))
nmask = fillHull(nmask)
nmask = bwlable(nmask)
if (interactive()) display(normalize(nmask), title='Cell nuclei mask')

## segment cells, using propagate and nuclei as 'seeds'
ctmask = opening(cel>0.1, makeBrush(5, shape='disc'))
cmask = propagate(cel, nmask, ctmask)
if (interactive()) display(normalize(cmask), title='Cell mask')

## using paintObjects to highlight objects
res = paintObjects(cmask, img, col='#ff00ff')
res = paintObjects(nmask, res, col='#ffff00')
if (interactive()) display(res, title='Segmented cells')

```

---

propagate

*Voronoi-based segmentation on image manifolds*


---

**Description**

Find boundaries between adjacent regions in an image, where seeds have been already identified in the individual regions to be segmented. The method finds the Voronoi region of each seed on a manifold with a metric controlled by local image properties. The method is motivated by the problem of finding the borders of cells in microscopy images, given a labelling of the nuclei in the images.

Algorithm and implementation are from Jones et al. [1].

**Usage**

```
propagate(x, seeds, mask=NULL, lambda=1e-4)
```

**Arguments**

x	An Image object or an array, containing the image to segment.
seeds	An Image object or an array, containing the seeding objects of the already identified regions.
mask	An optional Image object or an array, containing the binary image mask of the regions that can be segmented. If missing, the whole image is segmented.

lambda            A numeric value. The regularisation parameter used in the metric, determining the trade-off between the Euclidian distance in the image plane and the contribution of the gradient of  $x$ . See details.

### Details

The method operates by computing a discretized approximation of the Voronoi regions for given seed points on a Riemann manifold with a metric controlled by local image features.

Under this metric, the infinitesimal distance  $d$  between points  $v$  and  $v+dv$  is defined by:

$$d^2 = ( (t(dv)*g)^2 + lambda*t(dv)*dv ) / (lambda + 1)$$

, where  $g$  is the gradient of image  $x$  at point  $v$ .

lambda controls the weight of the Euclidian distance term. When lambda tends to infinity,  $d$  tends to the Euclidian distance. When lambda tends to 0,  $d$  tends to the intensity gradient of the image.

The gradient is computed on a neighborhood of 3x3 pixels.

Segmentation of the Voronoi regions in the vicinity of flat areas (having a null gradient) with small values of lambda can suffer from artefacts coming from the metric approximation.

### Value

An Image object or an array, containing the labelled objects.

### License

The implementation is based on CellProfiler C++ source code [2, 3]. An LGPL license was granted by Thouis Jones to use this part of CellProfiler's code for the propagate function.

### Author(s)

The original CellProfiler code is from Anne Carpenter <carpenter@wi.mit.edu>, Thouis Jones <thouis@csail.mit.edu>, In Han Kang <intheek@mit.edu>. Responsible for this implementation: Greg Pau.

### References

- [1] T. Jones, A. Carpenter and P. Golland, "Voronoi-Based Segmentation of Cells on Image Manifolds", CVBIA05 (535-543), 2005
- [2] A. Carpenter, T.R. Jones, M.R. Lamprecht, C. Clarke, I.H. Kang, O. Friman, D. Guertin, J.H. Chang, R.A. Lindquist, J. Moffat, P. Golland and D.M. Sabatini, "CellProfiler: image analysis software for identifying and quantifying cell phenotypes", Genome Biology 2006, 7:R100
- [3] CellProfiler: <http://www.cellprofiler.org>

### See Also

[bwlabel](#), [watershed](#)

**Examples**

```

## a paraboloid mountain in a plane
n = 400
x = (n/4)^2 - matrix(
  (rep(1:n, times=n) - n/2)^2 + (rep(1:n, each=n) - n/2)^2,
  nrow=n, ncol=n)
x = normalize(x)

## 4 seeds
seeds = array(0, dim=c(n,n))
seeds[51:55, 301:305] = 1
seeds[301:305, 101:105] = 2
seeds[201:205, 141:145] = 3
seeds[331:335, 351:355] = 4

lambda = 10^seq(-8, -1, by=1)
segmented = Image(dim=c(dim(x), length(lambda)))

for(i in seq(along=lambda)) {
  prop = propagate(x, seeds, lambda=lambda[i])
  prop = prop/max(prop)
  segmented[:,i] = prop
}

if(interactive()){
  display(x, title='Image')
  display(seeds/max(seeds), title='Seeds')
  display(segmented, title="Voronoi regions")
}

```

---

 resize

*Spatial linear transformations*


---

**Description**

The following functions perform all spatial linear transforms: reflection, rotation, translation, resizing, and general affine transform.

**Usage**

```

flip(x)
flop(x)
rotate(x, angle, filter="bilinear", output.dim, output.origin=c(0, 0))
translate(x, v, filter="none", output.dim)
resize(x, w, h, filter="bilinear", output.dim, output.origin=c(0, 0))
affine(x, m, filter=c("bilinear", "none"), output.dim)

```

**Arguments**

x                    An Image object or an array.  
 angle                A numeric specifying the image rotation angle in degrees.

filter	A character string indicating the interpolating sampling filter. Valid values are 'none' or 'bilinear'. See Details.
output.dim	A vector of 2 numbers indicating the dimension of the output image. Default is <code>dim(x)</code> in all transforms except in <code>resize</code> , where it is <code>c(w, h)</code> .
output.origin	A vector of 2 numbers indicating the output coordinates of the origin in pixels. Default is <code>c(0, 0)</code> .
v	A vector of 2 numbers denoting the translation vector in pixels.
w, h	Width and height of the resized image. One of these arguments can be missing to enable proportional resizing.
m	A 3x2 matrix describing the affine transformation. See Details.

### Details

`flip` transforms `x` in its vertical mirror image by reflecting the pixels around the central `x`-axis.

`flop` transforms `x` in its horizontal mirror image by reflecting the pixels around the central `y`-axis.

`rotate` rotates the image counter-clockwise with the specified angle, centered on the input image center. Rotation center is changed by modifying the argument `output.origin`.

`resize` resizes the image `x` to desired dimensions. Resizing center is changed by modifying the argument `output.origin`. Zooming, without changing the output dimension, is achieved by setting the arguments `w` and `h` to values different from `output.dim`.

`affine` returns the affine transformation of `x`, where pixels coordinates, denoted by the matrix `px`, are transformed to `cbind(px, 1)%*%m`.

All spatial transforms excepted `flip` and `flop` are based on the general affine transformation. Spatial interpolation could be of two type: `none`, also called nearest neighbor, where interpolated pixel value is computed by taking the closest pixel, or `bilinear`, where interpolated pixel is computed by bilinear approximation of the 4 neighboring pixels. The bilinear filter gives the smoother results.

### Value

An Image object or an array, containing the transformed version of `x`.

### Author(s)

Gregoire Pau, 2012

### See Also

[transpose](#)

### Examples

```
x <-readImage(system.file("images", "lena.png", package="EBImage"))
if (interactive()) display(x)

y <- flip(x)
if (interactive()) display(y, title='flip(x)')

y = flop(x)
if (interactive()) display(y, title='flop(x)')

y <- resize(x, 128)
```

```

if (interactive()) display(y, title='resize(x, 128)')

y <- rotate(x, 30)
if (interactive()) display(y, title='rotate(x, 30)')

y <- translate(x, c(120, -20))
if (interactive()) display(y, title='translate(x, c(120, -20))')

m <- matrix(c(0.6, 0.2, 0, -0.2, 0.3, 300), nrow=3)
if (interactive()) display(affine(x, m), title='affine transform')

```

---

rmObjects

*Object removal and reindexation*


---

### Description

The rmObjects functions deletes objects from an image by setting their pixel intensity values to 0. reenumerate re-enumerates all objects in an image from 0 (background) to the actual number of objects.

### Usage

```

rmObjects(x, index)

reenumerate(x)

```

### Arguments

x	An Image object in Grayscale color mode or an array containing object masks. Object masks are sets of pixels with the same unique integer value.
index	A numeric vector (or a list of vectors if x contains multiple frames) containing the indexes of objects to remove in the frame.

### Value

An Image object or an array, containing the new objects.

### Author(s)

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2006-2007

### See Also

[bwlabel](#), [watershed](#)

### Examples

```

## make objects
x = readImage(system.file('images', 'shapes.png', package='EBImage'))
x = x[110:512,1:130]
y = bwlabel(x)
if (interactive()) display(normalize(y), title='Objects')

```

```

## remove and reenumerate
y = rmObjects(y, 5)
if (interactive()) display(normalize(y), title='Removal')
y = reenumerate(y)
if (interactive()) display(normalize(y), title='Reenumerated')

```

---

stackObjects

*Places detected objects into an image stack*


---

### Description

Places detected objects into an image stack.

### Usage

```
stackObjects(x, ref, combine=TRUE, bg.col='black', ext)
```

### Arguments

x	An Image object or an array containing object masks. Object masks are sets of pixels with the same unique integer value.
ref	An Image object or an array, containing the intensity values of the objects.
combine	If x contains multiple images, specifies if the resulting list of image stacks with individual objects should be combined using combine into a single image stack.
bg.col	Background pixel color.
ext	A numeric controlling the size of the output simage. If missing, ext is estimated from data. See details.

### Details

stackObjects creates a set of nbobj images of size  $(2*ext+1, 2*ext+1)$ , where nbobj is the number of objects in x, and places each object of x in this set.

If not specified, ext is estimated using the 95% quantile of  $2*\sqrt{g.l1}$ , where g.l1 is the semi-major axis descriptor extracted from computeFeatures.moment, taken over all the objects of the image x.

### Value

An Image object containing the stacked objects contained in x. If x contains multiple images and if combine is TRUE, stackObjects returns a list of Image objects.

### Author(s)

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2006-2007

### See Also

[combine](#), [tile](#), [computeFeatures.moment](#)



**Examples**

```

## simple example
x = readImage(system.file('images', 'shapes.png', package='EBImage'))
x = x[110:512,1:130]
y = bwlabel(x)
if (interactive()) display(normalize(y), title='Objects')
z = stackObjects(y, normalize(y))
if (interactive()) display(z, title='Stacked objects')

## load images
nuc = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
cel = readImage(system.file('images', 'cells.tif', package='EBImage'))
img = rgbImage(green=cel, blue=nuc)
if (interactive()) display(img, title='Cells')

## segment nuclei
nmask = thresh(nuc, 10, 10, 0.05)
nmask = opening(nmask, makeBrush(5, shape='disc'))
nmask = fillHull(bwlabel(nmask))

## segment cells, using propagate and nuclei as 'seeds'
ctmask = opening(cel>0.1, makeBrush(5, shape='disc'))
cmask = propagate(cel, nmask, ctmask)

## using paintObjects to highlight objects
res = paintObjects(cmask, img, col='#ff00ff')
res = paintObjects(nmask, res, col='#ffff00')
if (interactive()) display(res, title='Segmented cells')

## stacked cells
st = stackObjects(cmask, img)
if (interactive()) display(st, title='Stacked objects')

```

---

thresh

*Adaptive thresholding*


---

**Description**

Thresholds an image using a moving rectangular window.

**Usage**

```
thresh(x, w=5, h=5, offset=0.01)
```

**Arguments**

x	An Image object or an array.
w, h	Width and height of the moving rectangular window.
offset	Thresholding offset from the averaged value.

**Details**

This function returns the binary image resulting from the comparison between an image and its filtered version with a rectangular window. It is equivalent of doing  $\{f = \text{matrix}(1, nc=2*w+1, nr=2*h+1) ; f=f/\text{sum}(f)\}$  but slightly faster. The function `filter2` provides hence more flexibility than `thresh`.

**Value**

An Image object or an array, containing the transformed version of `x`.

**Author(s)**

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2005-2007

**See Also**

`filter2`

**Examples**

```
x = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
if (interactive()) display(x)
y = thresh(x, 10, 10, 0.05)
if (interactive()) display(y)
```

---

tile

*Tiling/untiling images*

---

**Description**

Given a sequence of frames, `tile` generates a single image with frames tiled. `untile` is the inverse function and divides an image into a sequence of images.

**Usage**

```
tile(x, nx=10, lwd=1, fg.col="#E4AF2B", bg.col="gray")
untile(x, nim, lwd=1)
```

**Arguments**

<code>x</code>	An Image object, an array or a list of these objects.
<code>nx</code>	The number of tiled images in a row.
<code>lwd</code>	The width of the grid lines between tiled images, can be 0.
<code>fg.col</code>	The color of the grid lines.
<code>bg.col</code>	The color of the background for extra tiles.
<code>nim</code>	A numeric vector of 2 elements for the number of images in both directions.

## Details

After object segmentation, `tile` is a useful addition to `stackObjects` to have an overview of the segmented objects.

## Value

An Image object or an array, containing the tiled/untiled version of `x`.

## Author(s)

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2006-2007

## See Also

[stackObjects](#)

## Examples

```
## make a set of blurred Lenas
lena = readImage(system.file("images", "lena-color.png", package="EBImage"))
x = resize(lena, 128, 128)
xt = list()
for (t in seq(0.1, 5, len=9)) xt=c(xt, list(gblur(x, s=t)))
xt = combine(xt)
if (interactive()) display(xt, title='Blurred Lenas')

## tile
xt = tile(xt, 3)
if (interactive()) display(xt, title='Tiled Lenas')

## untile
xu = untile(lena, c(3, 3))
if (interactive()) display(xu, title='Lena blocks')
```

---

transpose

*Image Transposition*

---

## Description

Transposes an image by swapping its first two, i.e., spatial dimensions.

## Usage

```
transpose(x, coerce = FALSE)
```

## Arguments

<code>x</code>	an Image object or an array.
<code>coerce</code>	controls the coercion of <code>x</code> . By default the output is of the same class as the input. If <code>coerce = TRUE</code> then <code>x</code> becomes coerced to an array.

**Details**

The transposition of an image is performed by swapping the X and Y indices of its array representation.

**Value**

An Image object or an array, containing x with its XY dimensions transposed. When `coerce = TRUE` the output is coerced to an array.

**Note**

`transpose` is particularly useful when converting between different representations of image data in which the X and Y dimensions are swapped. Typically, in such context only the actual pixel data matters. For performance reasons it is best practice to issue the function directly on an Image object with `coerce = TRUE` rather than to extract its image data first and only then perform the transposition, or to transpose the Image object and coerce it to an array afterwards.

**Author(s)**

Andrzej Oles, <andrzej.oles@gmail.com>, 2012

**See Also**

[flip](#), [flop](#), [rotate](#)

**Examples**

```
x = readImage(system.file("images", "lena-color.png", package="EBImage"))
y = transpose(x)
if (interactive()) {
  display(x, title='Lena')
  display(y, title='Transposed Lena')
}

## performing the transposition of an image twice should result in the original image
z = transpose(y)
identical(x, z)
```

---

watershed

*Watershed transformation and watershed based object detection*

---

**Description**

Watershed transformation and watershed based object detection.

**Usage**

```
watershed(x, tolerance=1, ext=1)
```

**Arguments**

x	An Image object or an array.
tolerance	The minimum height of the object in the units of image intensity between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbors, which is the highest. Tolerance should be chosen according to the range of x. Default value is 1, which is a reasonable value if x comes from distmap.
ext	Radius of the neighborhood in pixels for the detection of neighboring objects. Higher value smoothes out small objects.

**Details**

The algorithm identifies and separates objects that stand out of the background (zero). After the water fill, the source image is flipped upside down and the resulting valleys (values with higher intensities) are filled in first until another object or background is met. The deepest valleys (pixels with highest intensity) become indexed first, starting from 1.

The function `bwlabel` is a simpler, faster alternative to segment connected objects from binary images.

**Value**

An Grayscale Image object or an array, containing the labelled version of x.

**Author(s)**

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2007

**See Also**

[bwlabel](#), [propagate](#)

**Examples**

```
x = readImage(system.file('images', 'shapes.png', package='EBImage'))
x = x[110:512,1:130]
if (interactive()) display(x, title='Binary')
y = distmap(x)
if (interactive()) display(normalize(y), title='Distance map')
w = watershed(y)
if (interactive()) display(normalize(w), title='Watershed')
```

# Index

- \*Topic **manip**
  - thresh, [33](#)
  - tile, [34](#)
  - watershed, [36](#)
- \*Topic **package**
  - EImage, [11](#)
  - EImage-deprecated, [13](#)
  - [,Image,ANY,ANY,ANY-method (Image), [18](#)
  - [,Image-method (Image), [18](#)
  - affine (resize), [29](#)
  - as.Image (Image), [18](#)
  - as.raster.Image (Image), [18](#)
  - blackTopHatGreyScale (morphology), [23](#)
  - blur, [15](#)
  - blur (EImage-deprecated), [13](#)
  - bwlabel, [2](#), [7](#), [14](#), [27](#), [28](#), [31](#), [37](#)
  - channel, [3](#)
  - closing (morphology), [23](#)
  - closingGreyScale (morphology), [23](#)
  - cmoments (EImage-deprecated), [13](#)
  - Color (Image), [18](#)
  - colorMode, [4](#)
  - colorMode (Image), [18](#)
  - colormode (Image), [18](#)
  - colorMode<- (Image), [18](#)
  - Combine, [4](#)
  - combine, [32](#)
  - combine (Combine), [4](#)
  - computeFeatures, [5](#)
  - computeFeatures.moment, [32](#)
  - convolve, [15](#)
  - dilate (morphology), [23](#)
  - dilateGreyScale (morphology), [23](#)
  - display, [8](#), [19](#), [21](#)
  - distmap, [9](#)
  - drawCircle, [10](#)
  - drawfont (EImage-deprecated), [13](#)
  - drawtext (EImage-deprecated), [13](#)
  - EImage, [11](#)
  - EImage-deprecated, [13](#)
  - edgeFeatures (EImage-deprecated), [13](#)
  - edgeProfile (EImage-deprecated), [13](#)
  - equalize (EImage-deprecated), [13](#)
  - erode (morphology), [23](#)
  - erodeGreyScale (morphology), [23](#)
  - fft, [15](#), [22](#)
  - fillHull, [14](#)
  - filter2, [15](#), [17](#)
  - flip, [36](#)
  - flip (resize), [29](#)
  - floodFill, [16](#)
  - flop, [36](#)
  - flop (resize), [29](#)
  - gblur, [17](#), [22](#)
  - getFeatures (EImage-deprecated), [13](#)
  - getFrame (Image), [18](#)
  - getNumberOfFrames (Image), [18](#)
  - Grayscale (Image), [18](#)
  - haralickFeatures (EImage-deprecated), [13](#)
  - haralickMatrix (EImage-deprecated), [13](#)
  - hist,Image-method (Image), [18](#)
  - hullFeatures (EImage-deprecated), [13](#)
  - Image, [5](#), [18](#), [21](#)
  - image,Image-method (Image), [18](#)
  - Image-class (Image), [18](#)
  - imageData (Image), [18](#)
  - imageData<- (Image), [18](#)
  - io, [20](#)
  - is.Image (Image), [18](#)
  - makeBrush, [15](#), [17](#), [22](#)
  - makeBrush (morphology), [23](#)
  - median.Image (Image), [18](#)
  - medianFilter, [22](#)
  - moments (EImage-deprecated), [13](#)
  - morphology, [23](#)
  - normalize, [25](#)
  - ocontour, [25](#)

opening (morphology), 23  
openingGreyScale (morphology), 23  
Ops,Image,Image-method (Image), 18  
Ops,Image,numeric-method (Image), 18  
Ops,numeric,Image-method (Image), 18

paintObjects, 26  
print.Image (Image), 18  
propagate, 7, 13, 27, 37

quantile.Image (Image), 18

readImage, 19  
readImage (io), 20  
readJPEG, 21  
readPNG, 21  
readTIFF, 21  
reenumerate (rmObjects), 31  
resize, 29  
rgbImage (channel), 3  
rmObjects, 31  
rmoments (EBImage-deprecated), 13  
rotate, 36  
rotate (resize), 29

selfcomplementaryTopHatGreyScale  
    (morphology), 23  
show,Image-method (Image), 18  
smoments (EBImage-deprecated), 13  
stackObjects, 32, 35  
standardExpandRef (computeFeatures), 5

thresh, 33  
tile, 32, 34  
translate (resize), 29  
transpose, 30, 35

until (tile), 34

watershed, 27, 28, 31, 36  
whiteTopHatGreyScale (morphology), 23  
writeImage, 19  
writeImage (io), 20  
writeJPEG, 21  
writePNG, 21  
writeTIFF, 20, 21

zernikeMoments (EBImage-deprecated), 13